Submitted, IEEE T. Automatic Control
http://www.cds.caltech.edu/~murray/papers/wtm10-tac.html

Receding Horizon Temporal Logic Planning

Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray

Abstract

We present a methodology for automatic synthesis of embedded control software that incorporates a class of linear temporal logic (LTL) specifications sufficient to describe a wide range of properties including safety, stability, progress, obligation, response and guarantee. To alleviate the associated computational complexity of LTL synthesis, we propose a receding horizon framework that effectively reduces the synthesis problem into a set of smaller problems. The proposed control architecture consists of a goal generator, a trajectory planner, and a continuous controller. The goal generator reduces the trajectory generation problem into a sequence of smaller problems of short horizon while preserving the desired system-level temporal properties. Subsequently, in each iteration, the trajectory planner solves the corresponding short-horizon problem with the currently observed state as the initial state and generates a feasible trajectory to be implemented by the continuous controller. Based on the simulation property, we show that the composition of the goal generator, trajectory planner and continuous controller and the corresponding receding horizon framework guarantee the correctness of the system with respect to its specification regardless of the environment in which the system operates. In addition, we present a response mechanism to handle failures that may occur due to a mismatch between the actual system and its model. The effectiveness of the proposed technique is demonstrated through an example of an autonomous vehicle navigating an urban environment. This example also illustrates that the system is not only robust with respect to exogenous disturbances but is also capable of properly handling violation of the environment assumption that is explicitly stated as part of the system specification .

Index Terms

Autonomous systems, control architecture, linear temporal logic, receding horizon control.

This work is partially supported by AFOSR and the Boeing Corporation.

Authors are with the Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125, USA. {nok, utopcu, murray}@cds.caltech.edu

I. INTRODUCTION

Design and verification of modern engineered systems with a tight link between computational and physical elements have become increasingly complex due to the interleaving between the high-level logics and the low-level dynamics. Consider, for example, an autonomous driving problem, particularly the 2007 DARPA Urban Challenge [1]. In this competition, all the competing vehicles had to navigate, in a fully autonomous manner, through a partially known urban-like environment populated with static and dynamic obstacles, including live traffic, and perform different tasks such as road and off-road driving, parking and visiting certain areas while obeying traffic rules. These tasks are specified by a sequence of checkpoints that the vehicle had to cross. For the vehicles to successfully complete the race, they need to be capable of negotiating an intersection, handling changes in the environment or operating condition (e.g. newly discovered obstacles) and reactively replanning in response to those changes (e.g. making a U-turn and finding a new route when the newly discovered obstacles fully block the road).

Alice, Team Caltech's entry in the DARPA Urban Challenge, is shown in Fig. 1. It was equipped with 25 CPUs and utilized a networked control system architecture to provide high performance and modular design. The planner-controller subsystem of Alice is shown in Fig. 1. This hierarchical planning architecture comprises the following modules [2], [3], [4]:

- Mission Planner computes the route, i.e., a sequence of roads the vehicle has to navigate in order to cross a given sequence of checkpoints. It is also capable of re-computing the route when the response from Traffic Planner indicates that the previously computed route cannot be navigated successfully. This type of failure occurs, for example, when the road is blocked.
- Traffic Planner makes decisions to guide Alice at a high level. Specifically, based on the traffic rules and the current environment, it determines how Alice should navigate the Mission Planner generated route, that is, whether it should stay in the travel lane or perform a passing maneuver, whether it should go or stop and whether it is allowed to reverse. In addition, it is responsible for intersection handling (e.g. keeping track of whether it is Alice's turn to go through an intersection). Based on these decisions, it sets up the constraints for the path planning problem.
- Path Planner generates a path that satisfies the constraints determined by Traffic Planner.

- Path Follower computes control signals (acceleration and steering angle) such that the vehicle closely follows the path generated by Path Planner.
- Gcdrive is the overall driving software for Alice. It contains a set of logics to protect the physical hardware and only allows valid actuation commands computed by Path Follower to be executed by the actuators. Examples of these hardware protection logics include limiting the steering rate at low speeds and preventing shifting from occuring while the vehicle is moving. Furthermore, Gcdrive implements the emergency stop functionality for Alice and stops the vehicle when an externally produced emergency stop command is received.

Fig. 1. *Left.* Alice, Team Caltech's entry in the 2007 DARPA Urban Challenge. *Right.* Alice's planner-controller subsystem.

This planner-controller subsystem was designed and implemented completely by hand in an ad-hoc manner. Furthermore, it was verified only through simulations and tests. Hence, there was absolutely no formal guarantee that the system would work as desired. Nevertheless, Alice successfully accomplished two of the three tasks at the National Qualifying Event (NQE). During the third task, which involved making left-turns while merging into traffic, its behavior was unsafe and almost led to a collision. The detailed analysis of the design flaw that led to this behavior can be found in [5]. In fact, this design flaw was partially known shortly before the second run of this particular test. However, it was difficult to modify and verify the design during the NQE due to the complexity of the system and the lack of sufficient time. Although it might be impossible to make such a system simple, part of the complexity could be avoided if the system had been designed in a systematic way.

Motivated by the difficulty of modifying and verifying complex systems such as Alice, in this paper, we investigate the problem of automatically synthesizing a planner-controller subsystem to provide a formal guarantee that, by construction, the system satisfies its specification expressed in linear temporal logic [6], [7], [8]. A common approach to such synthesis problem is to construct a finite transition system that serves as an abstract model of the physical system (which typically has infinitely many states) [9]–[17]. Then based on this abstract model, synthesize a strategy, represented by a finite state automaton, satisfying the specification. This leads to a hierarchical, two-layer design with a discrete planner computing a discrete plan based on the abstract model and a continuous controller computing a sequence of control signals based on the physical model to continuously implement the plan. Simulations/bisimulations [18] provide the proof that the continuous execution preserves the correctness of the discrete plan.

The correctness of this hierarchical approach relies on the abstraction of systems evolving on a continuous domain into equivalent (in the simulation sense) finite state models. If the abstraction is done properly such that the continuous controller is capable of implementing any discrete plan computed by the discrete planner, then it is guaranteed that the correctness of the plan is preserved in the continuous execution.

Several abstraction methods have been proposed based on a fixed abstraction. For example, a continuous-time, time-invariant model was considered in [10], [11] and [12] for special cases of fully actuated ($\dot{s}(t) = u(t)$), kinematic ($\dot{s}(t) = A(s(t))u(t)$) and piecewise affine (PWA) dynamics, respectively, while a discrete-time, time-invariant model was considered in [15] and [13] for special cases of PWA and controllable linear systems respectively. Reference [14] deals with more general dynamics by relaxing the bisimulation requirement and using the notions of approximate simulation and simulation functions [19]. More recently, a sampling-based method has been proposed for μ-calculus specifications [9]. However, these approaches do not take into account the presence of exogenous disturbances and the resulting system may fail to satisfy its specification if its evolution does not exactly match its model.

Another challenge of this hierarchical approach that remains an open problem and has received less attention in literature is the computational complexity in the synthesis of finite state automata. In particular, the synthesis problem becomes significantly harder when the interaction with the (potentially dynamic and not a priori known) environment has to be taken into account. Piterman et al. [20] treated this problem as a two-player game between the system and the

environment and proposed an algorithm for the synthesis of a finite state automaton that satisfies its specification regardless of the environment in which it operates (subject to certain assumptions on the environment that need to be stated in the specification). Although for a certain class of properties, known as *Generalized Reactivity[1]*, such an automaton can be computed in polynomial time, the applications of the synthesis tool are limited to small problems due to the state explosion issue.

Similar computational complexity is also encountered in the area of constrained optimal control. In the controls domain, an effective and well-established technique to address this problem is to design and implement control strategies in a receding horizon manner, i.e., optimize over a *shorter* horizon, starting from the currently observed state, implement the initial control action, move the horizon one step ahead, and re-optimize. This approach reduces the computational complexity by essentially solving a sequence of *smaller* optimization problems, each with a specific initial condition (as opposed to optimizing with *any* initial condition in traditional optimal control). Under certain conditions, receding horizon control strategies are known to lead to closed-loop stability [21], [22], [23]. See, e.g., [24] for a detailed discussion on constrained optimal control, including finite horizon optimal control and receding horizon control.

This paper concerns both the abstraction and the computational complexity issues. The remainder of the paper is organized as follows. In Section II, we present the key definitions and notations. The planner-controller synthesis problem is formulated in Section III. The hierarchical approach is described in detail in Section IV. To increase the robustness of the system against the effects of direct, external disturbances and a mismatch between the actual system and its model, in Section V, we provide an approach to automatically compute a finite state abstraction for a discrete-time linear time-invariant system, taking into account exogenous disturbances.

To reduce the complexity of the synthesis problem, in Section VI, we propose the receding horizon framework for executing finite state automata while ensuring system correctness with respect to a given linear temporal logic specification. The proposed framework allows the synthesis problem to be reduced to a set of smaller problems of short horizon. Its implementation, presented in Section VII, leads to the decomposition of the discrete planner into a goal generator and a trajectory planner. The goal generator reduces the synthesis problem to a sequence of short horizon problems while preserving the desired system-level temporal properties. Subsequently, in each iteration, the trajectory planner solves the corresponding short-horizon problem with the

currently observed state as the initial state and generates a feasible trajectory to be implemented by the continuous controller. Observe how this design corresponds to Alice's planner-controller subsystem with the goal generator having similar functionality as Mission Planner, the trajectory planner having similar functionality as the composition of Traffic Planner and Path Planner, and the continuous controller having similar functionality as Path Follower. Also presented in Section VII is a response mechanism that potentially increases the robustness of the system with respect to a mismatch between the actual system and its model and violation of the environment assumptions. Finally, in Section VIII, we demonstrate the effectiveness of the proposed technique through an example of an autonomous vehicle navigating an urban environment. This example also illustrates that the system is not only robust with respect to exogenous disturbances but also capable of handling violation of the environment assumptions.

Preliminary versions of this work have appeared in [15], [16], [17].

II. Preliminaries

We use linear temporal logic (LTL) to specify properties of systems. In this section, we first give formal definitions of terminology and notations used throughout the paper. Then, based on these definitions, we briefly describe LTL and some important classes of LTL formulas.

Definition 1. A system consists of a set V of variables. The *domain* of V, denoted by $dom(V)$, is the set of valuations of V. A *state* of the system is an element $v \in dom(V)$.

We describe an *execution* of a system by an infinite sequence of its states. Specifically, for a discrete-time system whose state is only evaluated at time $t \in \{0, 1, \ldots\}$, its execution σ can be written as $\sigma = v_0 v_1 v_2 \ldots$ where for each $t \geq 0$, $v_t \in dom(V)$ is the state of the system at time t.

Definition 2. A *finite transition system* is a tuple $\mathbb{T} := (\mathcal{V}, \mathcal{V}_0, \rightarrow)$ where $\mathcal{V}$ is a finite set of states, $\mathcal{V}_0 \subseteq \mathcal{V}$ is a set of inital states, and $\rightarrow \subseteq \mathcal{V} \times \mathcal{V}$ is a transition relation. Given states $\nu_i, \nu_j \in \mathcal{V}$, we write $\nu_i \rightarrow \nu_j$ if there is a transition from ν_i to ν_j in $\mathbb{T}$.

Observe that in this paper, we use ν to represent a state of a finite transition system and v to represent a state of a general, possibly non-finite state system.

Definition 3. A *partially ordered set* $(V, \preceq)$ consists of a set V and a binary relation $\preceq$ over the set V satisfying the following properties: for any $v_1, v_2, v_3 \in V$, (a) $v_1 \preceq v_1$; (b) if $v_1 \preceq v_2$ and $v_2 \preceq v_1$, then $v_1 = v_2$; and (c) if $v_1 \preceq v_2$ and $v_2 \preceq v_3$, then $v_1 \preceq v_3$.

Definition 4. An *atomic proposition* is a statement on system variables v that has a unique truth value ($True$ or $False$) for a given value of v. Let $v \in dom(V)$ be a state of the system and p be an atomic proposition. We write $v \Vdash p$ if p is $True$ at the state v. Otherwise, we write $v \nVdash p$.

Linear temporal logic [6], [7], [8] is a powerful specification language for unambiguously and concisely expressing a wide range of properties of systems. LTL is built up from (a) a set of atomic propositions, (b) the logic connectives: negation ($\neg$), disjunction ($\vee$), conjunction ($\wedge$) and material implication ($\Longrightarrow$), and (c) the temporal modal operators: next ($\bigcirc$), always ($\Box$), eventually ($\Diamond$) and until ($\mathcal{U}$). An LTL formula is defined inductively as follows: (1) any atomic proposition p is an LTL formula; and (2) given LTL formulas φ and ψ, $\neg\varphi$, $\varphi \vee \psi$, $\bigcirc\varphi$ and $\varphi \,\mathcal{U}\, \psi$ are also LTL formulas. Other operators can be defined as follows: $\varphi \wedge \psi \triangleq \neg(\neg\varphi \vee \neg\psi)$, $\varphi \Longrightarrow \psi \triangleq \neg\varphi \vee \psi$, $\Diamond\varphi \triangleq True\,\mathcal{U}\,\varphi$, and $\Box\varphi \triangleq \neg\Diamond\neg\varphi$. A *propositional formula* is one that does not include temporal operators. Given a set of LTL formulas $\varphi_1, \ldots, \varphi_n$, their *Boolean combination* is an LTL formula formed by joining $\varphi_1, \ldots, \varphi_n$ with logic connectives.

Semantics of LTL: An LTL formula is interpreted over an infinite sequence of states. Given an execution $\sigma = v_0 v_1 v_2 \ldots$ and an LTL formula φ, we say that φ *holds at position* $i \geq 0$ of σ, written $v_i \vDash \varphi$, if and only if (iff) φ holds for the remainder of the execution σ starting at position i. The semantics of LTL is defined inductively as follows: (a) For an atomic proposition p, $v_i \vDash p$ iff $v_i \Vdash p$; (b) $v_i \vDash \neg\varphi$ iff $v_i \nvDash \varphi$; (c) $v_i \vDash \varphi \vee \psi$ iff $v_i \vDash \varphi$ or $v_i \vDash \psi$; (d) $v_i \vDash \bigcirc\varphi$ iff $v_{i+1} \vDash \varphi$; and (e) $v_i \vDash \varphi \,\mathcal{U}\, \psi$ iff there exists $j \geq i$ such that $v_j \vDash \psi$ and $\forall k \in [i, j), v_k \vDash \varphi$. Based on this definition, $\bigcirc\varphi$ holds at position v_i iff φ holds at the next state v_{i+1}, $\Box\varphi$ holds at position i iff φ holds at every position in σ starting at position i, and $\Diamond\varphi$ holds at position i iff φ holds at some position $j \geq i$ in σ.

Definition 5. An execution $\sigma = v_0 v_1 v_2 \ldots$ *satisfies* φ, denoted by $\sigma \vDash \varphi$, if $v_0 \vDash \varphi$.

Definition 6. Let Σ be the set of all executions of a system. The system is said to be *correct* with respect to its specification φ, written $\Sigma \vDash \varphi$, if all its executions satisfy φ, that is, $(\Sigma \vDash \varphi)$ iff $\big(\forall\sigma, (\sigma \in \Sigma) \implies (\sigma \vDash \varphi)\big)$.

Examples of LTL formulas: Given propositional formulas p and q describing the states of interest, important and widely-used properties can be defined in terms of their corresponding LTL formulas as follows.

(a) **Safety** (invariance): A safety formula is of the form $\Box p$, which asserts that the property p remains invariantly true throughout an execution. Typically, a safety property ensures that nothing bad happens. A classic example of safety property frequently used in the robot motion planning domain is obstacle avoidance.

(b) **Guarantee** (reachability): A guarantee formula is of the form $\Diamond p$, which guarantees that the property p becomes true at least once in an execution. Reaching a goal state is an example of a guarantee property.

(c) **Obligation**: An obligation formula is a disjunction of safety and guarantee formulas, $\Box p \vee \Diamond q$. It can be shown that any safety and progress property can be expressed using an obligation formula. (By letting $q \equiv \mathit{False}$, we obtain a safety formula and by letting $p \equiv \mathit{False}$, we obtain a guarantee formula.)

(d) **Progress** (recurrence): A progress formula is of the form $\Box\Diamond p$, which essentially states that the property p holds infinitely often in an execution. As the name suggests, a progress property typically ensures that the system makes progress throughout an execution.

(e) **Response**: A response formula is of the form $\Box(p \implies \Diamond q)$, which states that following any point in an execution where the property p is true, there exists a point where the property q is true. A response property can be used, for example, to describe how the system should react to changes in the operating conditions.

(f) **Stability** (persistence): A stability formula is of the form $\Diamond \Box\, p$, which asserts that there is a point in an execution where the property p becomes invariantly true for the remainder of the execution. This definition corresponds to the definition of stability in the controls domain since it ensures that eventually, the system converges to a desired operating point and remains there for the remainder of the execution.

Remark 1. Properties typically studied in the control and hybrid systems domains are safety (usually in the form of constraints on the system state) and stability (i.e., convergence to an equilibrium or a desired state). LTL thus offers extensions to properties that can be expressed. Not only can it express a more general class of properties, but it also allows more general safety and stability properties than constraints on the system state or convergence to an equilibrium since p in $\Box p$ and $\Diamond \Box\, p$ can be any propositional formula.

III. Problem Formulation

We consider a system that comprises the physical component, which we refer to as the plant, and the (potentially dynamic and not a priori known) environment in which the plant operates. Assuming that the system specification φ is expressed in LTL, we are interested in automatically synthesizing a planner-controller subsystem that generates control signals to the plant in order to ensure that φ is satisfied in the presence of exogenous disturbances for any initial condition and any environment in which the plant operates. Specifically, we define the system model $\mathbb{S}$ and the specification φ as follows.

System Model: Consider a system model $\mathbb{S}$ with a set $V = S \cup E$ of variables where S and E are disjoint sets that represent, respectively, the set of plant variables that are regulated by the planner-controller subsystem and the set of environment variables whose values may change arbitrarily throughout an execution. The domain of V is given by $dom(V) = dom(S) \times dom(E)$ and a state of the system can be written as $v = (s, e)$ where $s \in dom(S) \subseteq \mathbb{R}^n$ and $e \in dom(E)$. Throughout the paper, we call s the *controlled* state and e the *environment* state.

Assume that the controlled state evolves according to the following discrete-time linear time-invariant state space model: for $t \in \{0, 1, 2, \ldots\}$,

$$\begin{aligned} s[t+1] &= As[t] + Bu[t] + Ed[t] \\ u[t] &\in U \\ d[t] &\in D \\ s[0] &\in dom(S) \end{aligned} \tag{1}$$

where $U \subseteq \mathbb{R}^m$ is the set of admissible control inputs, $D \subseteq \mathbb{R}^p$ is the set of exogenous disturbances and $s[t]$, $u[t]$ and $d[t]$ are the controlled state, the control signal, and the exogenous disturbance, respectively, at time t.

Example 1. Consider a robot motion planning problem where a robot needs to navigate an environment populated with (potentially dynamic) obstacles and explore certain areas of interest. S typically includes the state (e.g. position and velocity) of the robot while E typically includes the positions of obstacles (which are generally not known a priori and may change over time). The evolution of the controlled state (i.e., the state of the robot) is governed by its equations of motion, which can be written in the form of (1) (after linearization, if necessary).

System Specification: We assume that the specification φ consists of the following components:

(a) the assumption φ_{init} on the initial condition of the system,

(b) the assumption φ_e on the environment, and

(c) the desired behavior φ_s of the system.

Specifically, we assume that φ can be written as

$$\varphi = \big(\varphi_{init} \wedge \varphi_e\big) \Longrightarrow \varphi_s. \tag{2}$$

Let Π be a finite set of atomic propositions of variables from V. Each of the atomic propositions in Π essentially captures the states of interest. We assume that the desired behavior φ_s is an LTL specification built from Π and can be expressed as a conjunction of safety, guarantee, obligation, progress, response and stability properties as follows

$$\begin{aligned} \varphi_s = \; & \textstyle\bigwedge_{j\in J_1} \Box p^s_{1,j} \wedge \bigwedge_{j\in J_2} \Diamond p^s_{2,j} \wedge \\ & \textstyle\bigwedge_{j\in J_3} (\Box p^s_{3,j} \vee \Diamond q^s_{3,j}) \wedge \bigwedge_{j\in J_4} \Box\Diamond p^s_{4,j} \wedge \\ & \textstyle\bigwedge_{j\in J_5} \Box(p^s_{5,j} \Longrightarrow \Diamond q^s_{5,j}) \wedge \bigwedge_{j\in J_6} \Diamond\Box\, p^s_{6,j}, \end{aligned} \tag{3}$$

where $J_1, \ldots, J_6$ are finite sets and for any i and j, $p^s_{i,j}$ and $q^s_{i,j}$ are propositional formulas of variables from V that are built from Π.

Furthermore, we assume that φ_{init} is a propositional formula built from Π and φ_e can be expressed as a conjunction of safety and justice requirements as follows

$$\varphi_e = \bigwedge_{i\in I_1} \Box p^e_{f,i} \wedge \bigwedge_{i\in I_2} \Box\Diamond p^e_{s,i}, \tag{4}$$

where $p^e_{f,i}$ and $p^e_{s,i}$ are propositional formulas built from Π and only contain variables from E.

Example 2. Consider the robot motion planning problem described in Example 1. Suppose the workspace of the robot is partitioned into cells $C_1, \ldots, C_M$ and the robot needs to explore (i.e., visit) the cells C_1 and C_2 infinitely often. In addition, we assume that one of the cells $C_1, \ldots, C_M$ may be occupied by an obstacle at any given time and this obstacle-occupied cell may change arbitrarily throughout an execution but infinitely often, C_1 and C_2 are not occupied. Let s and o represent the position of the robot and the obstacle, respectively. In this case, the desired behavior of the system can be written as

$$\begin{aligned} \varphi_s = \; & \Box\Diamond(s \in C_1) \wedge \Box\Diamond(s \in C_2) \wedge \Box\big((o \in C_1) \Longrightarrow (s \notin C_1)\big) \wedge \\ & \Box\big((o \in C_2) \Longrightarrow (s \notin C_2)\big) \wedge \ldots \wedge \Box\big((o \in C_M) \Longrightarrow (s \notin C_M)\big). \end{aligned}$$

Assuming that initially, the robot does not occupy the same cell as the obstacle, we simply let $\varphi_{init} = \big((o \in C_1) \implies (s \notin C_1)\big) \wedge \big((o \in C_2) \implies (s \notin C_2)\big) \wedge \ldots \wedge \big((o \in C_m) \implies (s \notin C_m)\big)$. Finally, the assumption on the environment can be expressed as $\varphi_e = \Box\Diamond(o \notin C_1) \wedge \Box\Diamond(o \notin C_2)$.

Planner-Controller Synthesis Problem: Given the system model $\mathbb{S}$ and the system specification φ, synthesize a planner-controller subsystem that generates a sequence of control signals $u[0], u[1], \ldots \in U$ to the plant to ensure that starting from any initial condition, φ is satisfied for any sequence of exogenous disturbances $d[0], d[1], \ldots \in D$ and any sequence of environment states.

Remark 2. We restrict φ_s and φ_e to be of the form (3) and (4), respectively, for the clarity of presentation. Our framework only requires that the specification (2) can be reduced to the form of equation (6), presented later.

Remark 3. The specification φ has to be satisfied for any initial condition and environment, including those that violate the assumptions φ_{init} and φ_e. However, according to (2), satisfying φ ensures that the system exhibits the desired behavior φ_s only when φ_{init} and φ_e are satisfied.

IV. Hierarchical Approach

As described in Section I, we follow a hierarchical approach to attack the Planner-Controller Synthesis Problem defined in Section III. First, we construct a finite transition system $\mathbb{D}$ (e.g. a Kripke structure) that serves as an abstract model of $\mathbb{S}$ (which typically has infinitely many states). With this abstraction, the problem is then decomposed into (1) synthesizing a discrete planner that computes a discrete plan satisfying the specification φ based on the abstract, finite-state model $\mathbb{D}$, and (2) designing a continuous controller that implements the discrete plan. The success of this abstraction-based approach thus relies on the following two critical steps:

(a) an abstraction of an infinite-state system into an equivalent (in the simulation sense) finite state model such that any discrete plan generated by the discrete planner can be implemented (i.e., *simulated*; see, for example, [25] for the exact definition of *simulation*) by the continuous controller, provided that the evolution of the controlled state satisfies (1), and

(b) synthesis of a discrete planner (i.e., a strategy), represented by a finite state automaton, that ensures the correctness of the discrete plan.

In Section V, we present an approach to handle step (a), assuming that the physical system is modeled as described in Section III. To handle step (b) and ensure the system correctness for any initial condition and environment, we apply the two-player game approach presented in [20] to synthesize a discrete planner as in [10], [15]. In summary, consider a class of LTL formulas of the form

$$\left(\psi_{init} \wedge \Box\psi_e \wedge \bigwedge_{i\in I_f} \Box\Diamond\psi_{f,i}\right) \Longrightarrow \left(\bigwedge_{i\in I_s} \Box\psi_{s,i} \wedge \bigwedge_{i\in I_g} \Box\Diamond\psi_{g,i}\right), \tag{5}$$

known as *Generalized Reactivity[1]* (GR[1]) formulas. Here, ψ_{init}, $\psi_{f,i}$ and $\psi_{g,i}$ are propositional formulas of variables from V; ψ_e is a Boolean combination of propositional formulas of variables from V and expressions of the form $\bigcirc\psi_e^t$ where ψ_e^t is a propositional formula of variables from E that describes the assumptions on the transitions of environment states; and $\psi_{s,i}$ is a Boolean combination of propositional formulas of variables from V and expressions of the form $\bigcirc\psi_s^t$ where ψ_s^t is a propositional formula of variables from V that describes the constraints on the transitions of controlled states. The approach presented in [20] allows checking the realizability of this class of specifications and synthesizing the corresponding finite state automaton to be performed in time $O(|\mathcal{V}|^3)$ where $|\mathcal{V}|$ is the size of the state space of the finite state abstraction $\mathbb{D}$ of the system. We refer the reader to [20] and references therein for a detailed discussion.

Proposition 1. *A specification of the form (2) can be reduced to a subclass of GR[1] formula of the form*

$$\left(\psi_{init} \wedge \Box\psi_e^e \bigwedge_{i\in I_f} \Box\Diamond\psi_{f,i}^e\right) \Longrightarrow \left(\bigwedge_{i\in I_s} \Box\psi_{s,i} \wedge \bigwedge_{i\in I_g} \Box\Diamond\psi_{g,i}\right), \tag{6}$$

where ψ_{init}, $\psi_{s,i}$ and $\psi_{g,i}$ are as defined above and ψ_e^e and $\psi_{f,i}^e$ are propositional formulas of variables from E.

Throughout the paper, we call the left hand side and the right hand side of (6) the "assumption" part and the "guarantee" part, respectively. The proof of Proposition 1 is based on the fact that all safety, guarantee, obligation and response properties are special cases of progress formulas $\Box\Diamond p$, provided that p is allowed to be a past formula [6]. Hence, these properties can be reduced to the guarantee part of (6) by introducing auxiliary Boolean variables. For example, a guarantee property $\Diamond p_{2,j}^s$ can be reduced to the guarantee part of (6) by introducing an auxiliary Boolean variable x, initialized to $p_{2,j}^s$. $\Diamond p_{2,j}^s$ can then be equivalently expressed as a conjunction of $\Box\big((x \vee p_{2,j}^s) \Longrightarrow \bigcirc x\big)$, $\Box\big(\neg(x \vee p_{2,j}^s) \Longrightarrow \bigcirc(\neg x)\big)$ and $\Box\Diamond x$. Obligation and response

properties can be reduced to the guarantee part of (6) using a similar idea. In addition, a stability property $\diamond\Box p^s_{6,j}$ can be reduced to the guarantee part of (6) by introducing an auxiliary Boolean variable y, initialized to $False$. $\diamond \Box\, p^s_{6,j}$ can then be equivalently expressed as a conjunction of $\Box(y \implies p^s_{6,j})$, $\Box(y \implies \bigcirc y)$, $\Box\big(\neg y \implies (\bigcirc y \vee \bigcirc(\neg y))\big)$ and $\Box\diamond y$. Note that these reductions lead to equivalent specifications. However, for the case of stability, the reduction may lead to an unrealizable specification even though the original specification is realizable. Roughly speaking, this is because the auxiliary Boolean variable y needs to make clairvoyant (prophecy), non-deterministic decisions. For other properties, the realizability remains the same after the reduction since the synthesis algorithm [20] is capable of handling past formulas. The detail of this discussion is beyond the scope of this paper and we refer the reader to [20] for more detailed discussion on the synthesis of GR[1] specification.

In Section VI, we describe an extension of traditional receding horizon control to incorporate linear temporal logic specification of the form (6) in order to reduce the computational complexity of the synthesis problem. Its implementation and a response mechanism that enables the system to handle certain failures and continue to exhibit a correct behavior are presented in Section VII.

V. Computing Finite State Abstraction

To construct a finite transition system $\mathbb{D}$ from the physical model $\mathbb{S}$, we first partition $dom(S)$ and $dom(E)$ into finite sets $\mathcal{S}$ and $\mathcal{E}$, respectively, of equivalence classes or cells such that the partition is *proposition preserving* [18]. Roughly speaking, a partition is said to be proposition preserving if for any atomic proposition $\pi \in \Pi$ and any states v_1 and v_2 that belong to the same cell in the partition, v_1 satisfies π iff v_2 also satisfies π. We denote the resulting discrete domain of the system by $\mathcal{V} = \mathcal{S} \times \mathcal{E}$. We call $v \in dom(V)$ a *continuous state* and $\nu \in \mathcal{V}$ a *discrete state* of the system. For a discrete state $\nu \in \mathcal{V}$, we say that ν satisfies an atomic proposition $\pi \in \Pi$, denoted by $\nu \Vdash_d \pi$, if and only if there exists a continuous state v contained in the cell labeled by ν such that v satisfies π. Given an infinite sequence of discrete states $\sigma_d = \nu_0\nu_1\nu_2\ldots$ and LTL formula φ built from Π, we say that φ holds at position $i \geq 0$ of σ_d, written $\nu_i \vDash_d \varphi$, if and only if φ holds for the remainder of σ_d starting at position i. With these definitions, the semantics of LTL for a sequence of discrete states can be derived from the general semantics of LTL [6], [7], [8].

Next, we need to determine the transition relations $\rightarrow$ of $\mathbb{D}$. In Section V-A, we use a variant

of the notion of *reachability* that is sufficient to guarantee that if a discrete controlled state ς_j is reachable from ς_i, the transition from ς_i to ς_j can be continuously *implemented* or *simulated* by a continuous controller. A computational scheme that provides a sufficient condition for reachability between two discrete controlled states and subsequently refines the state space partition is also presented in Sections V-B and V-C.

A. Finite Time Reachability

Let $\mathcal{S} = \{\varsigma_1, \varsigma_2, \ldots, \varsigma_l\}$ be a set of discrete controlled states. We define a map $T_s : dom(S) \to \mathcal{S}$ that sends a continuous controlled state to a discrete controlled state of its equivalence class. That is, $T_s^{-1}(\varsigma_i) \subseteq dom(S)$ is the set of all the continuous controlled states contained in the cell labeled by ς_i and $\{T_s^{-1}(\varsigma_i), \ldots, T_s^{-1}(\varsigma_n)\}$ is the partition of $dom(S)$. We define the reachability relation, denoted by $\rightsquigarrow$, as follows.

Definition 7. A discrete state ς_j is *reachable* from a discrete state ς_i, written $\varsigma_i \rightsquigarrow \varsigma_j$, only if starting from any point $s[0] \in T_s^{-1}(\varsigma_i)$, there exists a horizon length $N \in \{0, 1, \ldots\}$ and a sequence of control signals $u[0], u[1], \ldots, u[N-1] \in U$ that takes the system (1) to a point $s[N] \in T_s^{-1}(\varsigma_j)$ satisfying the constraint $s[t] \in T_s^{-1}(\varsigma_i) \cup T_s^{-1}(\varsigma_j), \forall t \in \{0, \ldots, N\}$ for any sequence of exogenous disturbances $d[0], d[1], \ldots, d[N-1] \in D$. We write $\varsigma_i \not\rightsquigarrow \varsigma_j$ if ς_j is not reachable from ς_i.

In general, for two discrete states ς_i and ς_j, verifying the reachability relation $\varsigma_i \rightsquigarrow \varsigma_j$ is hard because it requires searching for a proper horizon length N. Therefore, we consider the restricted case where the horizon length is fixed and given and U, D and $T_s^{-1}(\varsigma_i), i \in \{1, \ldots, l\}$ are polyhedral sets. Our approach relies on solving the following problem.

Reachability Problem: Given an initial continuous controlled state $s[0] \in \mathbb{R}^n$, discrete controlled states $\varsigma_i, \varsigma_j \in \mathcal{S}$, the set of admissible control inputs $U \subseteq \mathbb{R}^m$, the set of exogenous disturbances $D \subseteq \mathbb{R}^p$, the matrices A, B and E as in (1), a horizon length $N \geq 0$, determine a sequence of control signals $u[0], u[1], \ldots, u[N-1] \in \mathbb{R}^m$ such that for all $t \in \{0, \ldots, N-1\}$ and $d[t] \in D$,

$$\begin{aligned} s[t+1] &= As[t] + Bu[t] + Ed[t], \\ s[t] &\in T_s^{-1}(\varsigma_i), \\ u[t] &\in U \\ s[N] &\in T_s^{-1}(\varsigma_j). \end{aligned} \tag{7}$$

B. Verifying the Reachability Relation

Given two discrete controlled states $\varsigma_i, \varsigma_j \in \mathcal{S}$, to determine whether $\varsigma_i \rightsquigarrow \varsigma_j$, we essentially have to verify that $T_s^{-1}(\varsigma_i) \subseteq S_0$ where S_0 is the set of $s[0]$ starting from which the Reachability Problem defined in Section V-A is feasible. In this section, we describe how S_0 can be computed using an idea from constrained robust optimal control [26].

We assume that U, D and $T_s^{-1}(\varsigma_i), i \in \{1, \ldots, l\}$ are polyhedral sets, i.e., there exist matrices L_1, L_2 and L_3 and vectors M_1, M_2 and M_3 such that $T_s^{-1}(\varsigma_i) = \{s \in \mathbb{R}^n \mid L_1 s \leq M_1\}$, $U = \{u \in \mathbb{R}^m \mid L_2 u \leq M_2\}$ and $T_s^{-1}(\varsigma_j) = \{s \in \mathbb{R}^n \mid L_3 s \leq M_3\}$. Then, by substituting

$$s[t] = A^t s[0] + \sum_{k=0}^{t-1} \left(A^k B u[t-1-k] + A^k E d[t-1-k] \right)$$

and replacing $s[t] \in T_s^{-1}(\varsigma_i)$, $u[t] \in U$ and $s[N] \in T_s^{-1}(\varsigma_j)$ with $L_1 s[t] \leq M_1$, $L_2 u[t] \leq M_2$ and $L_3 s[N] \leq M_3$, respectively, in (7), it can be easily checked that (7) can be rewritten in the form

$$L \begin{bmatrix} s[0] \\ \hat{u} \end{bmatrix} \leq M - G\hat{d}, \tag{8}$$

where $\hat{u} \triangleq [u[0]', \ldots, u[N-1]']' \in \mathbb{R}^{mN}$, $\hat{d} \triangleq [d[0]', \ldots, d[N-1]']' \in D^N$ and the matrices $L \in \mathbb{R}^{r \times n+mN}$ and $G \in \mathbb{R}^{r \times pN}$ and the vector $M \in \mathbb{R}^r$ can be obtained from L_1, L_2, L_3, M_1, M_2, M_3, A, B and E.

Using properties of polyhedral convexity, we can prove the following result.

Theorem 1. *Suppose D is a closed and bounded polyhedral subset of $\mathbb{R}^p$ and $\overline{D}$ is the set of all its extreme points. Let $P \triangleq \{y \in \mathbb{R}^{n+mN} \mid Ly \leq M - G\hat{d}, \forall \hat{d} \in \overline{D}^N\}$ and let S_0 be the projection of P onto its first n coordinates, i.e.,*

$$S_0 = \left\{ s \in \mathbb{R}^n \;\middle|\; \exists \hat{u} \in \mathbb{R}^{mN} \text{ s.t. } L \begin{bmatrix} s \\ \hat{u} \end{bmatrix} \leq M - G\hat{d}, \forall \hat{d} \in \overline{D}^N \right\}.$$

Then, the Reachability Problem defined in Section V-A is feasible for any $s[0] \in S_0$.

Using Theorem 1, the problem of computing the set S_0 such that the Reachability Problem is feasible for any $s[0] \in S_0$ is reduced to computing a projection of the intersection of finite sets and can be automatically solved using off-the-shelf software, for example, the Multi-Parametric Toolbox (MPT) [27].

C. State Space Discretization and Correctness of the System

In general, given the previous partition of $dom(S)$ and any $i, j \in \{1, \ldots, n\}$, the reachability relation between ς_i and ς_j may not be established through the set S_0 of $s[0]$ starting from which the Reachability Problem defined in Section V-A is feasible since $T_s^{-1}(\varsigma_i)$ is not necessarily covered by S_0 (due to the constraints on u and a specific choice of the finite horizon N). To partially alleviate this limitation, we refine the partition based on the reachability relation defined earlier to increase the number of valid discrete state transitions of $\mathbb{D}$. The underlying idea is that starting with an arbitrary pair of ς_i and ς_j, we determine the set S_0 of feasible $s[0]$ for the Reachability Problem. Then, we partition $T_s^{-1}(\varsigma_i)$ into $T_s^{-1}(\varsigma_i) \cap S_0$, labeled by $\varsigma_{i,1}$, and $T_s^{-1}(\varsigma_i) \backslash S_0$, labeled by $T_s^{-1}(\varsigma_{i,2})$, and obtain the following reachability relations: $\varsigma_{i,1} \rightsquigarrow \varsigma_j$ and $\varsigma_{i,2} \not\rightsquigarrow \varsigma_j$. This process is continued until some pre-specified termination criteria are met. Table I shows the pseudo-code of the algorithm where a prescribed lower bound Vol_{min} on the volume of each cell in the new partition is used as a termination criterion. The algorithm terminates when no cell can be partitioned such that the volumes of the two resulting new cells are both greater than Vol_{min}. Larger Vol_{min} causes the algorithm to terminate sooner. Other termination criteria such as the maximum number of iterations can be used as well. Note that the point at which the algorithm terminates affects the reachability between discrete controlled states of the new partition and as a result, affects the realizability of the specification. Generally, a coarse partition makes the specification unrealizable but a fine partition causes state space explosion. A way to decide when to terminate the algorithm is to start with a coarse partition and keep refining it until the specification is realizable.

We denote the set of all the discrete controlled states corresponding to the resulting partition of $dom(S)$ after applying the discretization algorithm by $\mathcal{S}'$. Since the partition obtained from the proposed algorithm is a refined partition of $\{T_s^{-1}(\mathcal{S}_1), \ldots, T_s^{-1}(\mathcal{S}_n)\}$ and $\mathcal{V} = \mathcal{S} \times \mathcal{E}$ is proposition preserving, it is trivial to show that $\mathcal{V}' = \mathcal{S}' \times \mathcal{E}$ is also proposition preserving. For simplicity of notation, we call $\mathcal{S}'$ as $\mathcal{S}$ and $\mathcal{V}'$ as $\mathcal{V}$ for the rest of the paper. We define the finite transition system $\mathbb{D}$ that serves as the abstract model of $\mathbb{S}$ as: $\mathcal{V} = \mathcal{S} \times \mathcal{E}$ is the set of states of $\mathbb{D}$ and for any two states $\nu_i = (\varsigma_i, \epsilon_i)$ and $\nu_j = (\varsigma_j, \epsilon_j)$, $\nu_i \rightarrow \nu_j$ (i.e., there exists a transition from ν_i to ν_j) only if $\varsigma_i \rightsquigarrow \varsigma_j$. Using the abstract model $\mathbb{D}$, a discrete planner that guarantees the satisfaction of φ while ensuring that the discrete plans are restricted to those satisfying the reachability relations can be automatically constructed using the digital design synthesis tool [20].

TABLE I

DISCRETIZATION ALGORITHM

Discretization Algorithm

input: The lower bound on cell volume (Vol_{min}), the parameters A, B, E, U, D, N of the Reachability Problem, and the original partition ($\{T_s^{-1}(\varsigma_i) \mid i \in \{1,\ldots,n\}\}$)

output: The new partition sol

$sol = \{T_s^{-1}(\varsigma_i) \mid i \in \{1,\ldots,n\}\}$; $IJ = \{(i,j) \mid i,j \in \{1,\ldots,n\}\}$;

while $(size(IJ) > 0)$

  Pick arbitrary ς_i and ς_j where $(i,j) \in IJ$;

  Compute the set S_0 of $s[0]$ starting from which the Reachability Problem is feasible for the previously chosen ς_i and ς_j;

  if $(volume(sol[i] \cap S_0) > Vol_{min}$ **and** $volume(sol[i] \backslash S_0 > Vol_{min})$ **then**

    Replace $sol[i]$ with $sol[i] \cap S_0$ and append $sol[i] \backslash S_0$ to sol;

    For each $k \in \{1,\ldots,size(sol)\}$, add (i,k), (k,i), $(size(sol),k)$ and $(k,size(sol))$ to IJ;

  else

    Remove (i,j) from IJ;

  endif

endwhile

From the stutter invariant property of φ [28], the formulation of the Reachability Problem and the proposition preserving property of $\mathcal{V}$, it is straightforward to prove the following proposition.

Proposition 2. *Let $\sigma_d = \nu_0\nu_1\ldots$ be an infinite sequence of discrete states of $\mathbb{D}$ where for each natural number k, $\nu_k \to \nu_{k+1}$, $\nu_k = (\varsigma_k, \epsilon_k)$, $\varsigma_k \in \mathcal{S}$ is the discrete controlled state and $\epsilon_k \in \mathcal{E}$ is the discrete environment state. If $\sigma_d \vDash_d \varphi$, then by applying a sequence of control signals, each corresponding to a solution of the Reachability Problem with $\varsigma_i = \varsigma_k$ and $\varsigma_j = \varsigma_{k+1}$, the infinite sequence of continuous states $\sigma = v_0v_1v_2\ldots$ satisfies φ.*

A solution $u[0],\ldots,u[N-1]$ of the Reachability Problem can be computed by formulating a constrained optimal control problem, which can be solved using off-the-shelf software such as MPT [27], YALMIP [29] or NTG [22].

VI. RECEDING HORIZON FRAMEWORK

The main limitation of the synthesis of finite state automata from their LTL specifications [20] is the state explosion problem. In the worst case, the resulting automaton may contain all the possible states of the system. For example, if the system has N variables, each can take any

value in $\{1, \ldots, M\}$, then there may be as many as M^N nodes in the automaton. This type of computational complexity limits the application of the systhesis to relatively small problems.

To reduce computational complexity in the synthesis of finite state automata, we apply an idea similar to the traditional receding horizon control [24]. First, we observe that in many applications, it is not necessary to plan for the whole execution, taking into account all the possible behaviors of the environment since a state that is very far from the current state of the system typically does not affect the near future plan. Consider, for example, the robot motion planning problem described in Example 2. Suppose C_1 or C_2 is very far away from the initial position of the robot. Under certain conditions, it may be sufficient to only plan out an execution for only a short segment ahead and implement it in a receding horizon fashion, i.e., re-compute the plan as the robot moves, starting from the currently observed state (rather than from all initial conditions satisfying φ_{init} as the original specification (2) suggests). In this section, we present a sufficient condition and a receding horizon strategy that allows the synthesis to be performed on a smaller domain; thus, substantially reduces the number of states (or nodes) of the automaton while still ensuring the system correctness with respect to the LTL specification (2).

We assume that a finite state abstraction $\mathbb{D}$ of the physical model $\mathbb{S}$ of the system has been constructed using, for example, the discretization algorithm presented in Section V-C. Let $\mathcal{V}$ be the finite set of states of $\mathbb{D}$. We consider a specification of the form (6) since, from Proposition 1, the specification (2) can be reduced to this form. Let Φ be a propositional formula of variables from V such that $\psi_{init} \implies \Phi$ is a tautology, i.e., any state $\nu \in \mathcal{V}$ that satisfies ψ_{init} also satisfies Φ. For each progress property $\Box\Diamond\psi_{g,i}$, $i \in I_g$, suppose there exists a collection of subsets $\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i$ of $\mathcal{V}$ such that

(a) $\mathcal{W}_0^i \cup \mathcal{W}_1^i \cup \ldots \cup \mathcal{W}_p^i = \mathcal{V}$,

(b) $\psi_{g,i}$ is satisfied for any $\nu \in \mathcal{W}_0^i$, i.e., $\mathcal{W}_0^i$ is the set of the states that constitute the progress of the system, and

(c) $\mathcal{P}^i := (\{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\}, \preceq_{\psi_{g,i}})$ is a partially ordered set defined such that $\mathcal{W}_0^i \prec_{\psi_{g,i}} \mathcal{W}_j^i, \forall j \neq 0$.

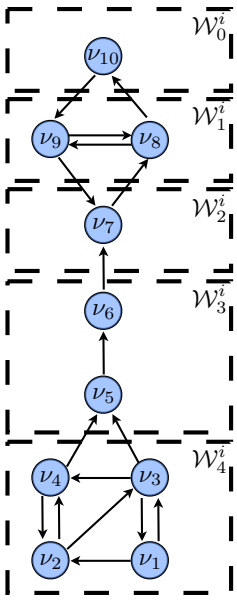

Fig. 2. Illustration of the receding horizon framework showing the relationships between the states of $\mathcal{V}$ and between the subsets $\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i$

Define a function $\mathcal{F}^i : \{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\} \to \{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\}$ such that $\mathcal{F}^i(\mathcal{W}_j^i) \prec_{\psi_{g,i}} \mathcal{W}_j^i, \forall j \neq 0$.

Consider a simple case where $\{\nu_1, \ldots, \nu_{10}\}$ is the set $\mathcal{V}$ of states, ν_{10} satisfies $\psi_{g,i}$, and the states in $\mathcal{V}$ are organized into 5 subsets $\mathcal{W}_0^i, \ldots, \mathcal{W}_4^i$. The relationships between the states in $\mathcal{V}$ and the subsets $\mathcal{W}_0^i, \ldots, \mathcal{W}_4^i$ are illustrated in Fig. 2. The partial order may be defined as $\mathcal{W}_0^i \prec \mathcal{W}_1^i \prec \ldots \prec \mathcal{W}_4^i$ and the mapping $\mathcal{F}^i$ may be defined as $\mathcal{F}^i(\mathcal{W}_j^i) = \mathcal{W}_{j-2}^i, \forall j \geq 2$, $\mathcal{F}^i(\mathcal{W}_1^i) = \mathcal{W}_0^i$ and $\mathcal{F}^i(\mathcal{W}_0^i) = \mathcal{W}_0^i$. Suppose ν_1 is the initial state of the system. The key idea of the receding horizon framework, as described later, is to plan from ν_1 to any state in $\mathcal{F}^i(\mathcal{W}_4^i) = \mathcal{W}_2^i$, rather than planning from the initial state ν_1 to the goal state ν_{10} in one shot, taking into account all the possible behaviors of the environment. Once a state in $\mathcal{W}_3^i$, i.e., ν_5 or ν_6 is reached, we then replan from that state to a state in $\mathcal{F}^i(\mathcal{W}_3^i) = \mathcal{W}_1^i$. We repeat this process until ν_{10} is reached. Under certain sufficient conditions presented later, this strategy ensures the correctness of the overall execution of the system.

Formally, with the above definitions of Φ, $\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i$ and $\mathcal{F}^i$, we define a short-horizon specification Ψ_j^i associated with $\mathcal{W}_j^i$ for each $i \in I_g$ and $j \in \{0, \ldots, p\}$ as

$$\begin{aligned} \Psi_j^i \;\triangleq\; & \Big((\nu \in \mathcal{W}_j^i) \wedge \Phi \wedge \Box\psi_e^e \wedge \textstyle\bigwedge_{k\in I_f} \Box\Diamond\psi_{f,k}^e\Big) \\ & \Longrightarrow \Big(\textstyle\bigwedge_{k\in I_s} \Box\psi_{s,k} \wedge \Box\Diamond\Big(\nu \in \mathcal{F}^i(\mathcal{W}_j^i)\Big) \wedge \Box\Phi\Big), \end{aligned} \tag{9}$$

where ν is the state of the system and ψ_e^e, $\psi_{f,k}^e$ and $\psi_{s,k}$ are defined as in (6).

An automaton $\mathcal{A}_j^i$ satisfying Ψ_j^i defines a strategy for going from a state $\nu_1 \in \mathcal{W}_j^i$ to a state $\nu_2 \in \mathcal{F}^i(\mathcal{W}_j^i)$ while satisfying the safety requirements $\bigwedge_{i\in I_s} \Box\psi_{s,i}$ and maintaining the invariant Φ (see Remark 5 for the role of Φ in this framework). Roughly speaking, the partial order $\mathcal{P}^i$ provides a measure of "closeness" to the states satisfying $\psi_{g,i}$. Since each specification Ψ_j^i asserts that the system eventually reaches a state that is smaller in the partial order, it ensures that each automaton $\mathcal{A}_j^i$ brings the system "closer" to the states satisfying $\psi_{g,i}$. The function $\mathcal{F}^i$ thus defines the horizon length for these short-horizon problems. In general, the size of $\mathcal{A}_j^i$ increases with the horizon length. However, with too short horizon, the specification Ψ_j^i is typically not realizable. A good practice is to choose the shortest horizon, subject to the realizability of Ψ_j^i, so that the automaton $\mathcal{A}_j^i$ contains as small number of states as possible.

Receding Horizon Strategy: For each $i \in I_g$ and $j \in \{0, \ldots, p\}$, construct an automaton $\mathcal{A}_j^i$ satisfying Ψ_j^i, defined in (9). Let $I_g = \{i_1, \ldots, i_n\}$ and define a corresponding ordered set $(i_1, \ldots, i_n)$. Note that this order only affects the sequence of progress properties $\psi_{g,i_1}, \ldots, \psi_{g,i_n}$

that the system tries to satisfy. Hence, it can be picked arbitrarily without affecting the correctness of the receding horizon strategy.

(1) Determine the index j_1 such that the current state $\nu_0 \in \mathcal{W}_{j_1}^{i_1}$. If $j_1 \neq 0$, then execute the automaton $\mathcal{A}_{j_1}^{i_1}$ until the system reaches a state $\nu_1 \in \mathcal{W}_k^{i_1}$ where $\mathcal{W}_k^{i_1} \prec_{\psi_{g,i_1}} \mathcal{W}_{j_1}^{i_1}$. Note that since the union of $\mathcal{W}_1^{i_1}, \ldots, \mathcal{W}_p^{i_1}$ is the set $\mathcal{V}$ of all the states, given any $\nu_0, \nu_1 \in \mathcal{V}$, there exist $j_1, k \in \{0, \ldots, p\}$ such that $\nu_0 \in \mathcal{W}_{j_1}^{i_1}$ and $\nu_1 \in \mathcal{W}_k^{i_1}$. This step corresponds to going from $\mathcal{W}_{j_1}^{i_1}$ to $\mathcal{W}_{j_1-1}^{i_1}$ in Fig. 3.

(2) If the current state $\nu_1 \notin \mathcal{W}_0^{i_1}$, switch to the automaton $\mathcal{A}_k^{i_1}$ where the index k is chosen such that the current state $\nu_1 \in \mathcal{W}_k^{i_1}$. Execute $\mathcal{A}_k^{i_1}$ until the system reaches a state that is smaller in the partial order $\mathcal{P}^{i_1}$. Repeat this step until a state $\nu_2 \in \mathcal{W}_0^{i_1}$ is reached. Note that it is guaranteed that a state $\nu_2 \in \mathcal{W}_0^{i_1}$ is eventually reached because of the finiteness of the set $\{\mathcal{W}_0^{i_1}, \ldots, \mathcal{W}_p^{i_1}\}$ and its partial order. See the proof of Theorem 2 for more details. This step corresponds to going all the way down the leftmost column in Fig. 3.

(3) Switch to the automaton $\mathcal{A}_{j_2}^{i_2}$ where the index j_2 is chosen such that the current state $\nu_2 \in \mathcal{W}_{j_2}^{i_2}$. Repeat step (2) with i_1 replaced by i_2 for the partial order $\mathcal{P}^{i_2}$ until a state $\nu_3 \in \mathcal{W}_0^{i_2}$ is reached. Repeat this step with i_2 replaced by $i_3, i_4, \ldots, i_n$, respectively, until a state $\nu_n \in \mathcal{W}_0^{i_n}$ is reached. In Fig. 3, this step corresponds to moving to the next column, going all the way down this column and repeating this process until we reach the bottom of the rightmost column.

(4) Repeat steps (1)–(3).

Theorem 2. *Suppose Ψ_j^i is realizable for each $i \in I_g$, $j \in \{0, \ldots, p\}$. Then the proposed receding horizon strategy ensures that the system is correct with respect to the specification (6), i.e., any execution of the system satisfies (6).*

Proof: Consider an arbitrary execution σ of the system that satisfies the assumption part of (6). We want to show that the safety properties $\psi_{s,i}, i \in I_s$, hold throughout the execution and for each $i \in I_g$, a state satisfying $\psi_{g,i}$ is reached infinitely often.

Let $\nu_0 \in \mathcal{V}$ be the initial state of the system and let the index j_1 be such that $\nu_0 \in \mathcal{W}_{j_1}^{i_1}$. From the tautology of $\psi_{init} \implies \Phi$, it is easy to show that σ satisfies the assumption part of $\Psi_{j_1}^{i_1}$ as defined in (9). Thus, if $j_1 = 0$, then $\mathcal{A}_0^{i_1}$ ensures that a state ν_2 satisfying ψ_{g,i_1} is eventually reached and the safety properties $\psi_{s,i}, i \in I_s$ hold at every position of σ up to and including the

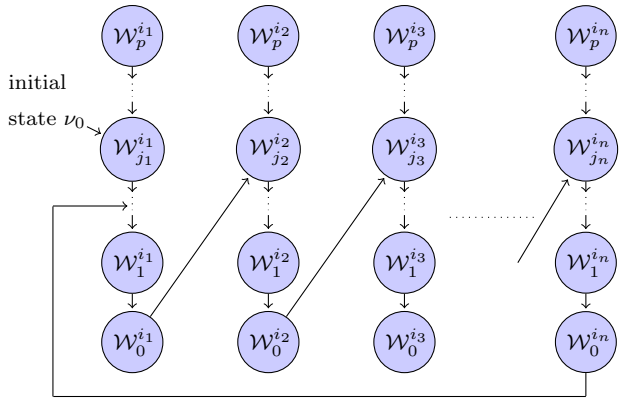

Fig. 3. A graphical description of the receding horizon strategy for a special case where for each $i \in I_g$, $\mathcal{W}_j^i \prec_{\psi_{g,i}} \mathcal{W}_k^i, \forall j < k$, $F^i(\mathcal{W}_j^i) = \mathcal{W}_{j-1}^i, \forall j > 0$ and $F^i(\mathcal{W}_0^i) = \mathcal{W}_0^i$. Starting from a state ν_0, the system executes the automaton $\mathcal{A}_{j_1}^{i_1}$ where the index j_1 is chosen such that $\nu_0 \in \mathcal{A}_{j_1}^{i_1}$. Repetition of step (2) ensures that a state $\nu_2 \in \mathcal{W}_0^{i_1}$ (i.e., a state satisfying ψ_{g,i_1}) is eventually reached. This state ν_2 belongs to some set, say, $\mathcal{W}_{j_2}^{i_2}$ in the partial order $\mathcal{P}^{i_2}$. The system then works through this partial order until a state $\nu_3 \in \mathcal{W}_0^{i_2}$ (i.e., a state satisfying ψ_{g,i_2}) is reached. This process is repeated until a state ν_n satisfying ψ_{g,i_n} is reached. At this point, for each $i \in I_g$, a state satisfying $\psi_{g,i}$ has been visited at least once in the execution. In addition, the state ν_n belongs to some set in the partial order $\mathcal{P}^{i_1}$ and the whole process is repeated, ensuring that for each $i \in I_g$, a state satisfying $\psi_{g,i}$ is visited infinitely often in the execution.

point where ν_2 is reached. Otherwise, $j_1 \neq 0$ and $\mathcal{A}_{j_1}^{i_1}$ ensures that eventually, a state $\nu_1 \in \mathcal{W}_k^{i_1}$ where $\mathcal{W}_k^{i_1} \prec_{\psi_g} \mathcal{W}_{j_1}^{i_1}$ is reached, i.e., ν_1 is the state of the system at some position l_1 of σ. In addition, the invariant Φ and all the safety properties $\psi_{s,i}, i \in I_s$, are guaranteed to hold at all the positions of σ up to and including the position l_1. According to the receding horizon strategy, the planner switches to the automaton $\mathcal{A}_k^{i_1}$ at position l_1 of σ. Since $\nu_1 \in \mathcal{W}_k^{i_1}$ and ν_1 satisfies Φ, the assumption part of $\Psi_k^{i_1}$ as defined in (9) is satisfied. Using the previous argument, we get that $\Psi_k^{i_1}$ ensures that all the safety properties $\psi_{s,i}, i \in I_s$, hold at every position of σ starting from position l_1 up to and including position l_2 at which the planner switches the automaton (from $\mathcal{A}_k^{i_1}$) and Φ holds at position l_2. By repeating this procedure and using the finiteness of the set $\{\mathcal{W}_0^{i_1}, \ldots, \mathcal{W}_p^{i_1}\}$ and its partial order condition, eventually the automaton $\mathcal{A}_0^{i_1}$ is executed which ensures that σ contains a state ν_2 satisfying ψ_{g,i_1} and step (2) terminates.

Applying the previous argument to step (3), we get that step (3) terminates and before it terminates, the safety properties $\psi_{s,i}, i \in I_s$, and the invariant Φ hold throughout the execution and for each $i \in I_g$, a state satisfying $\psi_{g,i}$ has been reached at least once. By continually repeating steps (1)–(3), the receding horizon strategy ensures that $\psi_{s,i}, i \in I_s$, hold throughout the execution

and for each $i \in I_g$, a state satisfying $\psi_{g,i}$ is reached infinitely often. ■

Remark 4. Traditional receding horizon control is known to not only reduce computational complexity but also increase the robustness of the system with respect to exogenous disturbances and modeling uncertainties [22]. With disturbances and modeling uncertainties, an actual execution of the system usually deviates from a reference trajectory s_d. Receding horizon control allows the current state of the system to be continually re-evaluated so s_d can be adjusted accordingly based on the externally received reference if the actual execution of the system does not match it. Such an effect may be expected in our extension of the traditional receding horizon control. Verifying this property is subject to current study.

Remark 5. The propositional formula Φ can be viewed as an invariant of the system. It adds an assumption on the initial state of each automaton $\mathcal{A}_j^i$ and is introduced in order to make Ψ_j^i realizable. Without Φ, the set of initial states of $\mathcal{A}_j^i$ includes all states $\nu \in \mathcal{W}_j^i$. However, starting from some "bad" state (e.g. unsafe state) in $\mathcal{W}_j^i$, there may not exist a strategy for the system to satisfy Ψ_j^i. Φ is essentially used to eliminate the possibility of starting from these "bad" states. Given a partially order set $\mathcal{P}^i$ and a function $\mathcal{F}^i$, one way to determine Φ is to start with $\Phi \equiv True$ and check the realizability of the resulting Ψ_j^i. If there exist $i \in I_g$ and $j \in \{0, \ldots, p\}$ such that Ψ_j^i is not realizable, the synthesis process provides the initial state ν^* of the system starting from which there exists a set of moves of the environment such that the system cannot satisfy Ψ_j^i. This information provides guidelines for constructing Φ, i.e., we can add a propositional formula to Φ that prevents the system from reaching the state ν^*. This procedure can be repeated until Ψ_j^i is realizable for any $i \in I_g$ and $j \in \{0, \ldots, p\}$ or until Φ excludes all the possible states, in which case either the original specification is unrealizable or the proposed receding horizon strategy cannot be applied with the given partially order set $\mathcal{P}^i$ and function $\mathcal{F}^i$.

Remark 6. For each $i \in I_g$ and $j \in \{0, \ldots, p\}$, checking the realizability of Ψ_j^i requires considering all the initial conditions in $\mathcal{W}_j^i$ satisfying Φ. However, as will be further discussed in Section VII, when a strategy (i.e., a finite state automaton satisfying Ψ_j^i) is to be extracted, only the currently observed state needs to be considered as the initial condition. Typically, the realizability can be checked symbolically and enumeration of states is only required when a

strategy needs to be extracted [20]. Symbolic methods are known to handle large number of states, in practice, significantly better than enumeration-based methods. Hence, state explosion usually occurs in the synthesis (i.e., strategy extraction) stage rather than the realizability checking stage. By considering only the currently observed state as the initial state in the synthesis stage, the receding horizon strategy delays state explosion both by considering a short-horizon problem and a specific initial state.

Remark 7. The proposed receding horizon approach is not complete. Even if there exists a control strategy that satisfies the original specification in (6), there may not exist an invariant Φ or a collection of subsets $\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i$ that allow the receding horizon strategy to be applied since the corresponding Ψ_j^i may not be realizable for all $i \in I_g$ and $j \in \{0, \ldots, p\}$.

VII. Implementation of the Receding Horizon Framework

In order to implement the receding horizon strategy described in Section VI, a partial order $\mathcal{P}^i$ and the corresponding function $\mathcal{F}^i$ need to be defined for each $i \in I_g$. In this section, we present an implementation of this strategy, allowing $\mathcal{P}^i$ and $\mathcal{F}^i$ to be automatically determined for each $i \in I_g$ while ensuring that all the short-horizon specifications $\Psi_j^i, i \in I_g, j \in \{0, \ldots, p\}$, as defined in (9) are realizable.

Given an invariant Φ and subsets $\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i$ of $\mathcal{V}$ for each $i \in I_g$, we first construct a finite transition system $\mathbb{T}^i$ with the set of states $\{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\}$. For each $j, k \in \{0, \ldots, p\}$, there is a transition $\mathcal{W}_j^i \to \mathcal{W}_k^i$ in $\mathbb{T}^i$ only if $j \neq k$ and the specification in (9) is realizable with $\mathcal{F}^i(\mathcal{W}_j^i) = \mathcal{W}_k^i$. The finite transition system $\mathbb{T}^i$ can be regarded as an abstraction of the finite state model $\mathbb{D}$ of the physical system $\mathbb{S}$, i.e., a higher-level abstraction of $\mathbb{S}$.

Suppose Φ is defined such that there exists a path in $\mathbb{T}^i$ from $\mathcal{W}_j^i$ to $\mathcal{W}_0^i$ for all $i \in I_g$, $j \in \{1, \ldots, p\}$. (Verifying this property is basically a graph search problem. If a path does not exist, Φ can be re-computed using a procedure described in Remark 5.) We propose a planner-controller subsystem with three components (cf. Fig. 4): goal generator, trajectory planner, and continuous controller.

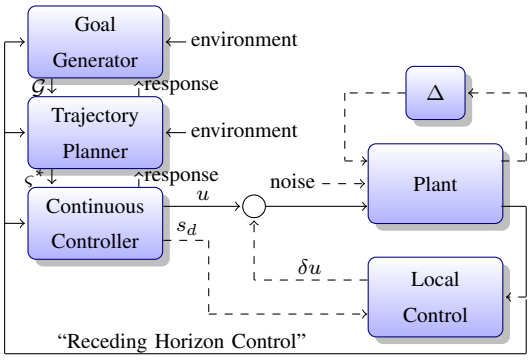

Fig. 4. A system with the planner-controller subsystem implemented in a receding horizon manner. In addition to the components discuss in this paper, Δ which captures uncertainties in the plant model may be added to make the model more realistic. In addition, the local control may be implemented to account for the effect of noise, disturbances, and unmodeled dynamics. The inputs and outputs of these two components are drawn in dashed since they are not considered in this paper.

Goal generator: Pick an order[1] $(i_1, \ldots, i_n)$ for the elements of the unordered set $I_g = \{i_1, \ldots, i_n\}$ and maintain an index $k \in \{1, \ldots, n\}$ throughout the execution. Starting with $k = 1$, in each iteration, the goal generator performs the following tasks.

(a1) Receive the currently observed state of the plant (i.e. the controlled state) and environment.

(a2) If the abstract state corresponding to the currently observed state belongs to $\mathcal{W}_0^{i_k}$, update k to $(k \mod n) + 1$.

(a3) If k was updated in step (a2) or this is the first iteration, then based on the higher level abstraction $\mathbb{T}^{i_k}$ of the physical system $\mathbb{S}$, compute a path from $\mathcal{W}_j^{i_k}$ to $\mathcal{W}_0^{i_k}$ where the index $j \in \{0, \ldots, p\}$ is chosen such that the abstract state corresponding to the currently observed state belongs to $\mathcal{W}_j^{i_k}$.

(a4) If a new path is computed in step (a3), then issue this path (i.e., a sequence $\mathcal{G} = \mathcal{W}_{l_0}^{i_k}, \ldots, \mathcal{W}_{l_m}^{i_k}$ for some $m \in \{0, \ldots, p\}$ where $l_0, \ldots l_m \in \{0, \ldots, p\}$, $l_0 = j$, $l_m = 0$, $l_\alpha \neq l_{\alpha'}$ for any $\alpha \neq \alpha'$, and there exists a transition $\mathcal{W}_{l_\alpha}^{i_k} \to \mathcal{W}_{l_{\alpha+1}}^{i_k}$ in $\mathbb{T}^{i_k}$ for any $\alpha < m$) to the trajectory planner.

Note that the problem of finding a path in $\mathbb{T}^{i_k}$ from $\mathcal{W}_j^{i_k}$ to $\mathcal{W}_0^{i_k}$ can be efficiently solved using any graph search or shortest-path algorithm [30], such as Dijkstra's, A*, etc. To reduce the original synthesis problem to a set of problems with short horizon, the cost on each edge $(\mathcal{W}_{l_\alpha}^{i_k}, \mathcal{W}_{l_{\alpha'}}^{i_k})$

[1]As discussed in the description of the receding horizon strategy in Section VI, this order can be picked arbitrarily. In general, its definition affects a strategy the system chooses to satisfy the specification (6) as it corresponds to the sequence of progress properties $\psi_{g,i_1}, \ldots, \psi_{g,i_n}$ the system tries to satisfy.

of the graph built from $\mathbb{T}^{i_k}$ may be defined, for example, as an exponential function of the "distance" between the sets $\mathcal{W}^{i_k}_{l_\alpha}$ and $\mathcal{W}^{i_k}_{l_{\alpha'}}$ so that a path with smaller cost contains segments of shorter "distance".

Trajectory planner: The trajectory planner maintains the latest sequence $\mathcal{G} = \mathcal{W}^{i_k}_{l_0}, \ldots, \mathcal{W}^{i_k}_{l_m}$ of goal states received from the goal generator, an index $q \in \{1, \ldots, m\}$ of the current goal state in $\mathcal{G}$, a strategy $\mathbb{F}$ represented by a finite state automaton, and the next abstract state ν^* throughout the execution. Starting with $q = 1$, $\mathbb{F}$ being an empty finite state automaton and ν^* being a null state, in each iteration, the trajectory planner performs the following tasks.

(b1) Receive the currently observed state of the plant and environment.

(b2) If a new sequence of goal states is received from the goal generator, update $\mathcal{G}$ to this latest sequence of goal states, update q to 1, and update ν^* to null. Otherwise, if the abstract state corresponding to the currently observed state belongs to $\mathcal{W}^{i_k}_{l_q}$, update q to $q + 1$ and ν^* to null.

(b3) If ν^* is null, then based on the abstraction $\mathbb{D}$ of the physical system $\mathbb{S}$, synthesize a strategy that satisfies the specification in (9) with $\mathcal{F}^i(\mathcal{W}^i_j) = \mathcal{W}^{i_k}_{l_q}$, starting from the abstract state ν_0 corresponding to the currently observed state, i.e., replace the assumption $\nu \in \mathcal{W}^i_j$ with $\nu = \nu_0$. Assign this strategy to $\mathbb{F}$ and update ν^* to the state following the initial state in $\mathbb{F}$ based on the current environment state.

(b4) If the controlled state ς^* component of ν^* corresponds to the currently observed state of the plant, update ν^* to the state following the current ν^* in $\mathbb{F}$ based on the current environment state.

(b5) If ν^* was updated in step (b3) or (b4), then issue the controlled state ς^* to the continuous controller.

Continuous controller: The continuous controller maintains the most recent (abstract) final controlled state ς^* from the trajectory planner. In each iteration, it receives the currently observed state s of the plant. Then, it computes a control signal u such that the continuous execution of the system eventually reaches the cell of $\mathbb{D}$ corresponding to the abstract controlled state ς^* while always staying in the cell corresponding to the abstract controlled state ς^* and the cell containing s. Essentially, the continuous execution has to *simulate* the abstract plan computed by the trajectory planner. As discussed at the end of Section V-C, such a control signal can

be computed by formulating a constrained optimal control problem, which can be solved using off-the-shelf software such as MPT [27], YALMIP [29] or NTG [22].

From the construction of $\mathbb{T}^i, i \in I_g$, it can be verified that the composition of the goal generator and the trajectory planner correctly implements the receding horizon strategy described in Section VI. Roughly speaking, the path $\mathcal{G}$ from $\mathcal{W}_j^i$ to $\mathcal{W}_0^i$ computed by the goal generator essentially defines the partial order $\mathcal{P}^i$ and the corresponding function $\mathcal{F}^i$. For a set $\mathcal{W}_{l_\alpha}^i \neq \mathcal{W}_0^i$ contained in $\mathcal{G}$, we simply let $\mathcal{W}_{l_{\alpha+1}}^i \prec \mathcal{W}_{l_\alpha}^i$ and $\mathcal{F}^i(\mathcal{W}_{l_\alpha}^i) = \mathcal{W}_{l_{\alpha+1}}^i$ where $\mathcal{W}_{l_{\alpha+1}}^i$ immediately follows $\mathcal{W}_{l_\alpha}^i$ in $\mathcal{G}$. In addition, since, by assumption, for any $i \in I_g$ and $l \in \{0, \ldots, p\}$, there exists a path in $\mathbb{T}^i$ from $\mathcal{W}_l^i$ to $\mathcal{W}_0^i$, it can be easily verified that the specification Ψ_l^i is realizable with $\mathcal{F}(\mathcal{W}_l^i) = \mathcal{W}_0^i$. Thus, to be consistent with the previously described receding horizon framework, we assign $\mathcal{W}_l^i \succ \mathcal{W}_0^i$ and $\mathcal{F}(\mathcal{W}_l^i) = \mathcal{W}_0^i$ for a set $\mathcal{W}_l^i$ not contained in $\mathcal{G}$. Note that such $\mathcal{W}_l^i$ that is not in the path $\mathcal{G}$ does not affect the computational complexity of the synthesis algorithm. With this definition of the partial order $\mathcal{P}^i$ and the corresponding function $\mathcal{F}^i$, we can apply Theorem 2 to conclude that the abstract plan generated by the trajectory planner ensures the correctness of the system with respect to the specification in (6). In addition, since the continuous controller *simulates* this abstract plan, the continuous execution is guaranteed to preserve the correctness of the system.

The resulting system is depicted in Fig. 4. Note that since it is guaranteed to satisfy the specification in (6), the desired behavior (i.e. the guarantee part of (6)) is ensured only when the environment and the initial condition respect their assumptions. To moderate the sensitivity to violation of these assumptions, the trajectory planner may send a response to the goal generator, indicating the failure of executing the last received sequence of goals as a consequence of assumption violation. The goal generator can then remove the problematic transition from the corresponding finite transition system $\mathbb{T}^i$ and re-compute a new sequence $\mathcal{G}$ of goals. This procedure will be illustrated in the example presented in Section VIII. Similarly, a response may be sent from the continuous controller to the trajectory planner to account for the mismatch between the actual system and its model. In addition, a local control may be added in order to account for the effect of the noise and unmodeled dynamics captured by Δ.

VIII. EXAMPLE

Motivated by the challenges faced in the design and verification of a DARPA Urban Challenge vehicle such as Alice, as described in Section I, we consider an autonomous vehicle navigating an urban environment while avoiding obstacles and obeying certain traffic rules. The state of the vehicle is the position (x, y) whose evolution is governed by

$$\dot{x}(t) = u_x(t) + d_x(t) \quad \text{and} \quad \dot{y}(t) = u_y(t) + d_y(t)$$

where $u_x(t)$ and $u_y(t)$ are control signals and $d_x(t)$ and $d_y(t)$ are external disturbances at time t. The control effort is subject the constraints $u_x(t), u_y(t) \in [-1, 1], \forall t \geq 0$. We assume that the disturbances are bounded by $d_x(t), d_y(t) \in [-0.1, 0.1], \forall t \geq 0$.

We consider the road network shown in Fig. 5 with 3 intersections, I_1, I_2 and I_3, and 6 roads, R_1, R_2 (joining I_1 and I_3), R_3, R_4 (joining I_2 and I_3), R_5 (joining I_1 and I_3) and R_6 (joining I_1 and I_2). Each of these roads has two lanes going in opposite directions. The positive and negative directions for each road are shown in Fig. 5. We partition the roads and intersections into $N = 282$ cells (cf. Fig. 5), each of which may be occupied by an obstacle.

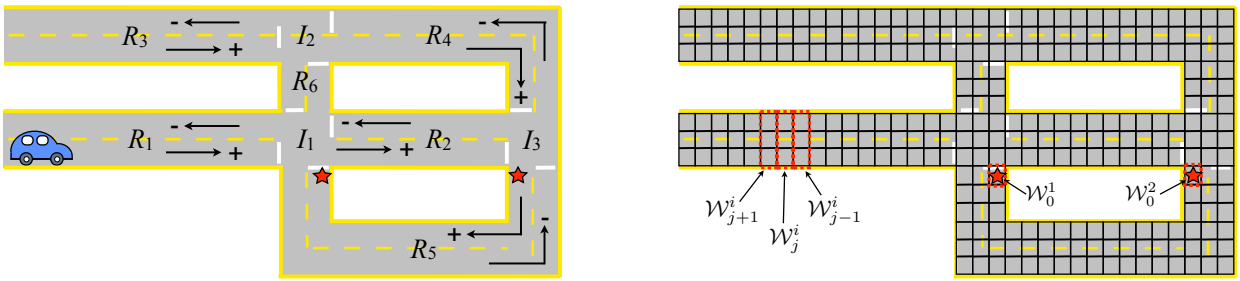

Fig. 5. The road network and its partition for the autonomous vehicle example. The solid (black) lines define the states in the set $\mathcal{V}$ of the finite state model $\mathbb{D}$ used by the trajectory planner. Examples of subsets $\mathcal{W}^i_j$ are drawn in dotted (red) rectangles. The stars indicate the positions that need to be visited infinitely often.

As described in Section I, a planner-controller subsystem of Alice is implemented in a hierarchical fashion with Mission Planner computing a route (i.e., a sequence of roads to be navigated) to achieve the given tasks, the composition of Traffic Planner and Path Planner computing a path (i.e., a sequence of desired positions) that describes how the vehicle should navigate the route generated by Mission Planner while satisfying the traffic rules, and Path Follower computing a control signal such that the vehicle closely follows the path generated by

Traffic Planner. Observe how this hierarchical approach naturally follows our general framework for designing a planner-controller subsystem (cf. Fig. 4) with Mission Planner being an instance of a goal generator and each of the sets $\mathcal{W}_1^i, \ldots, \mathcal{W}_p^i$ being an entire road. However, these components are typically designed by hand and validated through extensive simulations and field tests. Although a correct-by-construction approach has been applied in [31], it is based on building a finite state abstraction of the physical system and synthesizing a planner that computes a strategy for the whole execution, taking into account all the possible behaviors of the environment. As discussed in Section IV, this approach fails to handle even modest size problems due to the state explosion issue. In this section, we apply the receding horizon scheme to substantially reduce computational complexity of the correct-by-construction approach.

Given this system model, we want to design a planner-controller subsystem for the vehicle based on the following desired behavior and assumptions.

Desired Behavior: Following the terminology and notations used in Section III, the desired behavior φ_s in (2) includes the following properties.

(P1) Each of the two cells marked by star needs to be visited infinitely often.

(P2) No collision, i.e., the vehicle cannot occupy the same cell as an obstacle.

(P3) The vehicle stays in the travel lane (i.e., right lane) unless there is an obstacle blocking the lane.

(P4) The vehicle can only proceed through an intersection when the intersection is clear.

Assumptions: We assume that the vehicle starts from an obstacle-free cell on R_1 with at least one obstacle-free cell adjacent to it. This constitutes the assumption φ_{init} on the initial condition of the system. The environment assumption φ_e encapsulates the following statements which are assumed to hold throughout each execution.

(A1) Obstacles may not block a road.

(A2) An obstacle is detected before the vehicle gets too close to it, i.e., an obstacle may not instantly pop up right in front of the vehicle.

(A3) Sensing range is limited, i.e., the vehicle cannot detect an obstacle that is away from it farther than certain distance. In this example, we let this sensing range be 2 cells ahead in the driving direction.

(A4) To make sure that the stay-in-lane property is achievable, we assume that an obstacle does

not disappear while the vehicle is in its vicinity.

(A5) Obstacles may not span more than a certain number of consecutive cells in the middle of the road.

(A6) Each of the intersections is clear infinitely often.

(A7) Each of the cells marked by star and its adjacent cells are not occupied by an obstacle infinitely often.

It can be shown [32] that the properties (P2) and (P3) and the assumptions (A1)–(A4) can be expressed in the form of the guarantee and the assumption parts of (6). Property (P4) can be expressed as a safety formula and property (P1) is a progress property. Finally, assumption (A5) can be expressed as a safety assumption on the environment while assumptions (A6) and (A7) can be expressed as justice requirements on the environment.

We follow the approach described in Section IV. First, we compute a finite state abstraction $\mathbb{D}$ of the system. Following the scheme in Section V, a state ν of $\mathbb{D}$ can be written as $\nu = (\varsigma, \rho, o_1, o_2, \ldots, o_M)$ where $\varsigma \in \{1, \ldots, M\}$ and $\rho \in \{+, -\}$ are the controlled state components of ν, specifying the cell occupied by the vehicle and the direction of travel, respectively, and for each $i \in \{1, \ldots, M\}$, $o_i \in \{0, 1\}$ indicates whether the i^{th} cell is occupied by an obstacle. This leads to the total of $2M2^M$ possible states of $\mathbb{D}$. With the horizon length $N = 12$, it can be shown that based on the Reachability Problem defined in Section V-A, there is a transition $\nu_1 \to \nu_2$ in $\mathbb{D}$ if the controlled state components of ν_1 and ν_2 correspond to adjacent cells (i.e., they share an edge in the road network of Fig. 5).

Since the only progress property is to visit the two cells marked by star infinitely often, the set I_g in (6) has two elements, say, $I_g = \{1, 2\}$. We let $\mathcal{W}_0^1$ be the set of abstract states whose ς component corresponds to one of these two cells and define $\mathcal{W}_0^2$ similarly for the other cell as shown in Fig. 5. Other $\mathcal{W}_j^i$ is defined such that it includes all the abstract states whose ς component corresponds to cells across the width of the road (cf. Fig. 5).

Next, we define Φ such that it excludes states where the vehicle is not in the travel lane while there is no obstacle blocking the lane and states where the vehicle is in the same cell as an obstacle or none the cells adjacent to the vehicle are obstacle-free. Using this Φ, the specification in (9) is realizable with $\mathcal{F}^i(\mathcal{W}_j^i) = \mathcal{W}_k^i$ where $\mathcal{W}_j^i$ and $\mathcal{W}_k^i$ correspond to adjacent dotted (red) rectangles in Fig. 5. The finite transition system $\mathbb{T}^i$ used by the goal planner can then be constructed such that there is a transition $\mathcal{W}_j^i \to \mathcal{W}_k^i$ for any adjacent $\mathcal{W}_j^i$ and $\mathcal{W}_k^i$. With this

transition relation, for any $i \in I_g$ and $j \in \{0, \ldots, p\}$, there exists a path in $\mathbb{T}^i$ from $\mathcal{W}_j^i$ to $\mathcal{W}_0^i$ and the trajectory planner essentially only has to plan one step ahead. Thus, the size of finite state automata synthesized by the trajectory planner to satisfy the specification in (9) is completely independent of M. Using JTLV [20], each of these automata has less than 900 states and only takes approximately 1.5 seconds to compute on a MacBook with a 2 GHz Intel Core 2 Duo processor. In addition, with an efficient graph search algorithm, the computation time requires by the goal generator is in the order of milliseconds. Hence, with a real-time implementation of optimization-based control such as NTG [33] at the continuous controller level, our approach can be potentially implemented in real-time.

A simulation result is shown in Fig. 6(a), illustrating a correct execution of the vehicle even in the presence of exogenous disturbances when all the assumptions on the environment and initial condition are satisfied. Note that without the receding horizon strategy, there can be as many as 10^{87} states in the automaton, making this problem practically impossible to solve.

To illustrate the benefit of the response mechanism, we add a road blockage on R_2 to violate the assumption (A1). The result is shown in Fig. 6(b). Once the vehicle discovers the road blockage, the trajectory planner cannot find the current state of the system in the finite state automaton synthesized from the specification in (9) since the assumption on the environment is violated. The trajectory planner then informs the goal generator of the failure to satisfy the corresponding specification with the associated pair of $\mathcal{W}_j^i$ and $\mathcal{F}(\mathcal{W}_j^i)$. Subsequently, the goal generator removes the transition from $\mathcal{W}_j^i$ to $\mathcal{F}(\mathcal{W}_j^i)$ in $\mathbb{T}^i$ and re-computes a path to $\mathcal{W}_0^i$. As a result, the vehicle continues to exhibit a correct behavior by making a U-turn and completing the task using a different path.

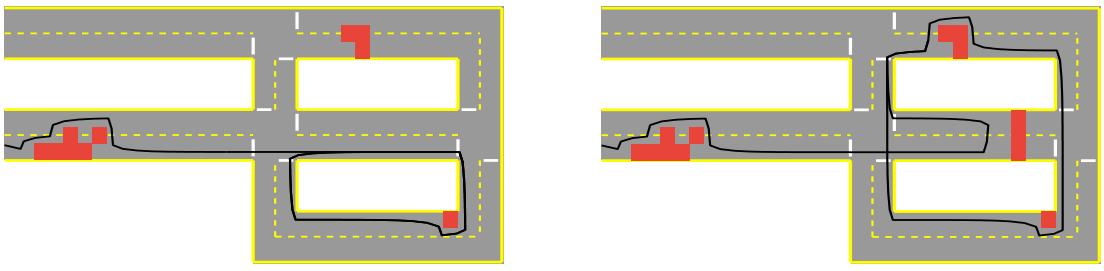

Fig. 6. Simulation results with (left) no road blockage, (right) a road blockage on R_2.

The result with exactly the same setup is also shown in Fig. 7 where the presence of exogenous disturbances is not incorporated in the planner-controller subsystem synthesis. Once the vehicle overtakes the obstacles on R_1, the continuous controller computes the sequence of control inputs that is expected to bring the vehicle back to its travel lane as commanded by the trajectory planner. However, due to the disturbance, the vehicle remains in the opposite lane. As a consequence, the trajectory planner keeps sending commands to the continuous controller to bring the vehicle back to its travel lane but even though the control inputs computed by the continuous controller are supposed to bring the vehicle back to its travel lane, the vehicle remains in the opposite lane due to the disturbance. In the meantime, the disturbance also causes the vehicle to drift slowly to the right. This cycle continues, leading to violation of the desired property that the vehicle has to stay in the travel lane unless there is an obstacle blocking the lane.

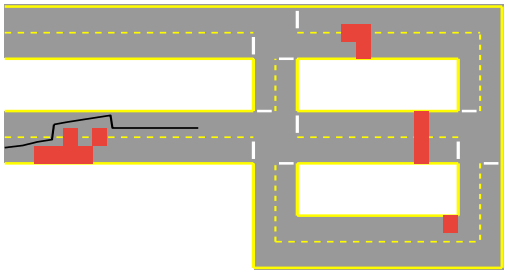

Fig. 7. Simulation result with the presence of disturbances not incorporated in the planner-controller subsystem synthesis.

IX. Conclusions and Future Work

Motivated by the DARPA Urban Challenge, we considered the planner-controller synthesis problem. Specifically, we proposed an approach to automatically synthesizing a planner-controller subsystem that ensures system correctness with respect to its specification expressed in linear temporal logic regardless of the environment in which the system operates. A receding-horizon-based framework that allows a computationally complex synthesis problem to be reduced to a set of significantly smaller problems was presented. An implementation of the proposed framework leads to a hierarchical, modular design with a goal generator, a trajectory planner and a continuous controller. A response mechanism that increases the robustness of the system with respect to a mismatch between the system and its model and between the actual behavior of the environment and its assumptions was discussed. By taking into account the presence of exogenous disturbances in the synthesis process, the resulting system is provably robust with respect to bounded exogenous disturbances.

Future work includes further investigation of the robustness of the receding horizon framework.

Specifically, we want to formally identify the types of properties and faults/failures that can be correctly handled using the proposed response mechanism. This type of mechanism has been implemented on Alice, an autonomous vehicle built at Caltech for the DARPA Urban Challenge, for distributed mission and contingency management [4]. Based on extensive simulations and field tests, it has been shown to handle many types of failures and faults at different levels of the system, including inconsistency of the states of different software modules and hardware and software failures.

Another direction of research is to study an asynchronous execution of the goal generator, the trajectory planner and the continuous controller. As described in this paper, these components are to be executed sequentially. However, with certain assumptions on the communication channels, a distributed, asynchronous implementation of these components may still guarantee the correctness of the system. Finally, we want to extend the proposed receding horizon framework to other class of temporal logics that allow better specification of temporal properties.

ACKNOWLEDGMENTS

The authors gratefully acknowledge Hadas Kress-Gazit for the inspiring discussions on the synthesis of finite state automata and Yaniv Sa'ar for the discussions on reducing stability properties to a GR[1] formula and his suggestions and help with JTLV, a framework in which the digital design synthesis tool is implemented.

REFERENCES

[1] "DARPA Urban Challenge," http://www.darpa.mil/ grandchallenge/index.asp, 2007.

[2] J. W. Burdick, N. DuToit, A. Howard, C. Looman, J. Ma, R. M. Murray, and T. Wongpiromsarn, "Sensing, navigation and reasoning technologies for the DARPA Urban Challenge," DARPA Urban Challenge Final Report, Tech. Rep., 2007.

[3] N. E. DuToit, T. Wongpiromsarn, J. W. Burdick, and R. M. Murray, "Situational reasoning for road driving in an urban environment," in *International Workshop on Intelligent Vehicle Control Systems (IVCS)*, 2008.

[4] T. Wongpiromsarn and R. M. Murray, "Distributed mission and contingency management for the DARPA Urban Challenge," in *International Workshop on Intelligent Vehicle Control Systems (IVCS)*, 2008.

[5] T. Wongpiromsarn, S. Mitra, R. M. Murray, and A. Lamperski, "Periodically controlled hybrid systems: Verifying a controller for an autonomous vehicle," in *Hybrid Systems: Computation and Control*, ser. LNCS, R. Majumdar and P. Tabuada, Eds., vol. 5469. Springer, 2009, pp. 396–410.

[6] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems*. Springer-Verlag, 1992.

[7] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*, 2nd ed. Cambridge University Press, 2004.

[8] E. A. Emerson, "Temporal and modal logic," in *Handbook of theoretical computer science (vol. B): formal models and semantics*. Cambridge, MA, USA: MIT Press, 1990, pp. 995–1072.

[9] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proc. of IEEE Conference on Decision and Control*, 2009.

[10] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *Proc. of IEEE International Conference on Robotics and Automation*, April 2007, pp. 3116–3121.

[11] D. Conner, H. Kress-Gazit, H. Choset, A. Rizzi, and G. Pappas, "Valet parking without a valet," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 572–577.

[12] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transaction on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[13] P. Tabuada and G. J. Pappas, "Linear time logic control of linear systems," *IEEE Transaction on Automatic Control*, vol. 51, no. 12, pp. 1862–1877, 2006.

[14] A. Girard and G. J. Pappas, "Hierarchical control system design using approximate simulation," *Automatica*, vol. 45, no. 2, pp. 566–571, 2009.

[15] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning for dynamical systems," in *Proc. of IEEE Conference on Decision and Control*, 2009.

[16] ——, "Automatic synthesis of robust embedded control software," in *AAAI Spring Symposium on Embedded Reasoning: Intelligence in Embedded Systems*, 2010, pp. 104–111.

[17] ——, "Receding horizon control for temporal logic specifications," in *Proc. of the 13th International Conference on Hybrid Systems: Computation and Control*, 2010.

[18] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971–984, 2000.

[19] A. Girard, A. A. Julius, and G. J. Pappas, "Approximate simulation relations for hybrid systems," *Discrete Event Dynamic Systems*, vol. 18, no. 2, pp. 163–179, 2008.

[20] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *Verification, Model Checking and Abstract Interpretation*, ser. Lecture Notes in Computer Science, vol. 3855. Springer-Verlag, 2006, pp. 364 – 380, software available at http://jtlv.sourceforge.net/.

[21] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, pp. 789–814, 2000.

[22] R. M. Murray, J. Hauser, A. Jadbabaie, M. B. Milam, N. Petit, W. B. Dunbar, and R. Franz, "Online control customization via optimization-based control," in *Software-Enabled Control: Information Technology for Dynamical Systems*. Wiley-Interscience, 2002, pp. 149–174, software available at http://www.cds.caltech.edu/~murray/software/2002a_ntg.html.

[23] A. Jadbabaie, "Nonlinear receding horizon control: A control Lyapunov function approach," Ph.D. dissertation, California Institute of Technology, 2000.

[24] G. C. Goodwin, M. M. Seron, and J. A. D. Doná, *Constrained Control and Estimation: An Optimisation Approach*. Springer, 2004.

[25] H. Tanner and G. J. Pappas, "Simulation relations for discrete-time linear systems," in *Proc. of the IFAC World Congress on Automatic Control*, 2002, pp. 1302–1307.

[26] F. Borrelli, *Constrained Optimal Control of Linear and Hybrid Systems*, ser. Lecture Notes in Control and Information Sciences. Springer, 2003, vol. 290.

[27] M. Kvasnica, P. Grieder, and M. Baotić, "Multi-Parametric Toolbox (MPT)," 2004, software available at http://control.ee.ethz.ch/~mpt/. [Online]. Available: http://control.ee.ethz.ch/~mpt/

[28] D. Peled and T. Wilke, "Stutter-invariant temporal properties are expressible without the next-time operator," *Inf. Process. Lett.*, vol. 63, no. 5, pp. 243–246, 1997.

[29] J. Löfberg, "YALMIP : A toolbox for modeling and optimization in MATLAB," in *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004, software available at http://control.ee.ethz.ch/~joloef/yalmip.php.

[30] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[31] H. Kress-Gazit and G. J. Pappas, "Automatically synthesizing a planning and control subsystem for the DARPA Urban Challenge," in *IEEE International Conference on Automation Science and Engineering*, 2008, pp. 766–771.

[32] T. Wongpiromsarn, "Formal methods for embedded control systems: Application to an autonomous vehicle," Ph.D. dissertation, California Institute of Technology, 2010, in preparation.

[33] M. B. Milam, K. Mushambi, and R. M. Murray, "A new computational approach to real-time trajectory generation for constrained mechanical systems," in *Proc. of IEEE Conference on Decision and Control*, 2000, pp. 845–851.