Receding Horizon Control for Temporal Logic Specifications

Tichakorn Wongpiromsarn
California Institute of Technology
Pasadena, CA
nok@caltech.edu

Ufuk Topcu
California Institute of Technology
Pasadena, CA
utopcu@cds.caltech.edu

Richard M. Murray
California Institute of Technology
Pasadena, CA
murray@cds.caltech.edu

ABSTRACT

In this paper, we describe a receding horizon framework that satisfies a class of linear temporal logic specifications sufficient to describe a wide range of properties including safety, stability, progress, obligation, response and guarantee. The resulting embedded control software consists of a goal generator, a trajectory planner, and a continuous controller. The goal generator essentially reduces the trajectory generation problem to 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. To handle failures that may occur due to a mismatch between the actual system and its model, we propose a response mechanism and illustrate, through an example, how the system is capable of responding to certain failures and continues to exhibit a correct behavior.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods*; D.2.10 [**Software Engineering**]: Design—*Methodologies*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Control theory; Plan execution, formation, and generation*

General Terms

Design, Verification

Keywords

Embedded control software, linear temporal logic, receding horizon control

1. INTRODUCTION

Synthesis of correct-by-construction embedded control software based on temporal logic specifications has attracted considerable attention in the recent years due to the increasing frequency of systems with tight integration between computational and physical elements and the complexity in designing and verifying such systems. A common approach is to construct a finite transition system that serves as an abstract model of the physical system (which typically has infinitely many states) and synthesize a strategy, represented by a finite state automaton, satisfying the given temporal properties based on this model. One of the main challenges of this approach is in the abstraction of systems evolving on a continuous domain into equivalent (in the simulation sense) finite state models. Several methods have been proposed based on a fixed abstraction for different cases of system dynamics [11, 2, 10, 22, 19, 5]. More recently, a sampling-based method has been proposed for μ-calculus specifications [9].

Another challenge that remains an open problem and has received less attention in literature is 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. [17] 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 strategy 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 [14, 16, 8]. See, for example, [6] for a detailed discussion on constrained optimal control, including finite horizon optimal control and receding horizon control.

In this paper, we build on our previous work [22] and further develop the receding horizon framework for executing finite state automata while ensuring system correctness with respect to a given temporal logic specification. This essentially allows the synthesis problem to be reduced to a set of smaller problems of short horizon. This paper is the enhanced version of [22] in several ways. First, we deal with a much richer class of linear temporal logic specifications. In [22], we only allow safety and guarantee properties, while in this paper we also deal with stability, progress, obligation and response, in addition to safety and guarantee properties. (See Section 2 for the exact definitions.) Second, we add an additional layer to the embedded control software, namely, a goal generator, in order to allow a sequence of short-horizon problems to be automatically generated (as opposed to having to manually generate these problems as in [22]). Finally, we present 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. The benefit of adding this mechanism is illustrated through an example where the system continues to exhibit a correct behavior even though some of the environment assumptions do not hold during an execution.

2. PRELIMINARIES

We use linear temporal logic (LTL) to describe the desired properties of the system. 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 $v \in dom(V)$.

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.

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 $v_1 = v_2$; (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 $v \nVdash p$.

Definition 5. An *execution* σ of a discrete-time system is an infinite sequence of the system states over a particular run, i.e., 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.

Linear Temporal Logic

Linear temporal logic (LTL) [13, 7, 3] is a powerful specification language for unambiguously and concisely expressing a wide range of properties of systems. LTL is built up from a set of atomic propositions, the logic connectives ($\neg$, $\vee$, $\wedge$, $\Longrightarrow$), and the temporal modal operators ($\bigcirc$, $\Box$, $\Diamond$, $\mathcal{U}$ which are read as "next," "always," "eventually," and "until," respectively). 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: $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$, $\varphi \Longrightarrow \psi = \neg\varphi \vee \psi$, $\Diamond\varphi = \textit{True}\ \mathcal{U}\ \varphi$, and $\Box\varphi = \neg\Diamond\neg\varphi$. 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 $v_i \models \sigma$, 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, $v_i \models p$ iff $v_i \Vdash p$; (b) $v_i \models \neg\varphi$ iff $v_i \not\models \varphi$; (c) $v_i \models \varphi \vee \psi$ iff $v_i \models \varphi$ or $v_i \models \psi$; (d) $v_i \models \bigcirc\varphi$ iff $v_{i+1} \models \varphi$; and (e) $v_i \models \varphi \,\mathcal{U}\, \psi$ iff $\exists j \geq i, v_j \models \psi$ and $\forall k \in [i, j), v_k \models \varphi$. 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 6. An execution $\sigma = v_0 v_1 v_2 \ldots$ *satisfies* φ, denoted by $\sigma \models \varphi$, if $v_0 \models \varphi$.

Definition 7. 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 $\Sigma \models \varphi$, if all its executions satisfy φ, that is, $(\Sigma \models \varphi) \iff \big(\forall\sigma, (\sigma \in \Sigma) \implies (\sigma \models \varphi)\big)$.

Examples: 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.

Safety (invariance): A safety formula is of the form $\Box p$, which simply 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.

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, i.e., a state satisfying p is reachable. Reaching a goal state is an example of a guarantee property.

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

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 keeps making progress throughout the execution.

Response: A response formula is of the form $\Box(p \Longrightarrow \Diamond q)$, 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 to describe how the system should react to changes in the operating conditions.

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 essentially 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 other classes 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.

3. PROBLEM FORMULATION

We are interested in designing embedded control software for a system that interacts with its (potentially dynamic and not a priori known) environment. This software needs to ensure that the system satisfies the desired property φ_s for any initial condition and any environment in which it operates, provided that the initial condition and the environment satisfy certain assumptions, φ_{init} and φ_e, respectively.

Specifically, we define the system model $\mathbb{S}$, the desired property φ_s and the assumptions φ_{init} and φ_e 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 the set of variables controlled by the system and the set of variables controlled by the environment respectively. 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 $s \in dom(S)$ 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 either a discrete-time, time-invariant dynamics

$$s(t+1) = f(s(t), u(t)), \quad u(t) \in U, \quad \forall t \geq 0 \tag{1}$$

or a continuous-time, time-invariant dynamics

$$\dot{s}(t) = f(s(t), u(t)), \quad u(t) \in U, \quad \forall t \in \mathbb{N} \tag{2}$$

where U is the set of admissible control inputs and $s(t)$ and $u(t)$ are the controlled state and control signal 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 prior and may change over time). The evolution of the controlled state (i.e., the state of the robot) is simply governed its equations of motion.

Desired Properties and Assumptions: 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 property 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:

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

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.

We further assume that the initial condition of the system satisfies a propositional formula φ_{init} built from Π and the environment satisfies an assumption φ_e where φ_e can be expressed as a conjunction of justice requirements and propositions that are assumed to be true throughout an execution as follows:

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

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

In summary, the specification of $\mathbb{S}$ is given by

$$\big(\varphi_{init} \;\wedge\; \varphi_e\big) \implies \varphi_s. \tag{5}$$

Observe, from the specification (5), that the desired property φ_s is guaranteed only when the assumptions on the initial condition and the environment are satisfied.

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

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 (5) can be reduced to the form of equation (7), presented later.

4. HIERARCHICAL APPROACH

A common approach (see, for example, [11, 2, 10, 22, 19, 5]) to designing embedded control software for a physical system $\mathbb{S}$ that provably satisfies a temporal logic specification is to construct a finite transition system $\mathbb{D}$ (e.g. 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 (a) synthesizing a planner that computes a discrete plan satisfying the specification based on the abstract, finite-state model $\mathbb{D}$, and (b) designing a continuous controller that implements the discrete plan. The success of this abstraction-based approach thus heavily 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 planner can be implemented (i.e., *simulated*; see, for example, [20] for the exact definition) by the continuous controller, and
(b) synthesis of a planner (i.e., a strategy), represented by a finite state automaton, that ensures the correctness of the discrete plan.

Different approaches have been proposed to handle step (a). For example, a continuous-time, time-invariant model (2) was considered in [11], [2] and [10] 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 (1) was considered in [22] and [19] for special cases of PWA and controllable linear systems respectively. Reference [5] deals with more general dynamics by relaxing the bisimulation requirement and using the notions of approximate simulation and simulation functions [4].

In this paper, we focus on addressing the computational complexity of step (b). We assume that a finite state abstraction $\mathbb{D}$ of the physical system $\mathbb{S}$ has been constructed. We denote the (finite) set of states of $\mathbb{D}$ by $\mathcal{V}$. In order to ensure the system correctness for any initial condition and environment, we apply the two-player game approach presented in [17] to synthesize a planner as in [11, 22]. In summary, consider a class of LTL formulas of the form

$$(\psi_{init} \wedge \square\psi_e \wedge \bigwedge_{i\in I_f} \square\diamond\psi_{f,i}) \implies (\bigwedge_{i\in I_s} \square\psi_{s,i} \wedge \bigwedge_{i\in I_g} \square\diamond\psi_{g,i}), \tag{6}$$

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 [17] allows checking the realizability of this class of specifications and synthesizing the corresponding finite state automaton to be performed in time $O(M^3)$ where M is the size of the state space of the system. We refer the reader to [17] and references therein for a detailed discussion.

Proposition 1. By introducing auxiliary Boolean variables, a specification of the form (5) can be reduced to a subclass of GR[1] formula of the form:

$$\big(\psi_{init} \wedge \square\psi_e^e \bigwedge_{i\in I_f} \square\diamond\psi_{f,i}^e\big) \implies \big(\bigwedge_{i\in I_s} \square\psi_{s,i} \wedge \bigwedge_{i\in I_g} \square\diamond\psi_{g,i}\big), \tag{7}$$

where ψ_{init}, $\psi_{s,i}$ and $\psi_{g,i}$ are defined as in (6) 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 (7) 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 $\square\diamond p$, provided that p is allowed to be a past formula [13]. (See [13] for the definition of a past formula and how each of these properties can be reduced to an instance of a progress property.) Hence, these properties can be reduced to the guarantee part of (7) by introducing auxiliary Boolean variables. For example, a guarantee property $\diamond p_{2,j}^s$ can be reduced to the guarantee part of (7) 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 $\square\big((x \vee p_{2,j}^s) \implies \bigcirc x\big)$, $\square\big(\neg(x \vee p_{2,j}^s) \implies \bigcirc(\neg x)\big)$ and $\square\diamond x$. Obligation and response properties can be reduced to the guarantee part of (7) using a similar idea. In addition, a stability property $\diamond\square p_{6,j}^s$ can be reduced to the guarantee part of (7) by introducing an auxiliary Boolean variable y, initialized to *False*. $\diamond\square p_{6,j}^s$ can then be equivalently expressed as a conjunction of $\square(y \implies p_{6,j}^s)$, $\square(y \implies \bigcirc y)$, $\square\big(\neg y \implies (\bigcirc y \vee \bigcirc(\neg y))\big)$ and $\square\diamond y$. 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 [17] is capable of handling past formulas. The detail of this discussion is beyond the scope of this paper and we refer the reader to [17] for more detailed discussion on the synthesis of GR[1] specification.

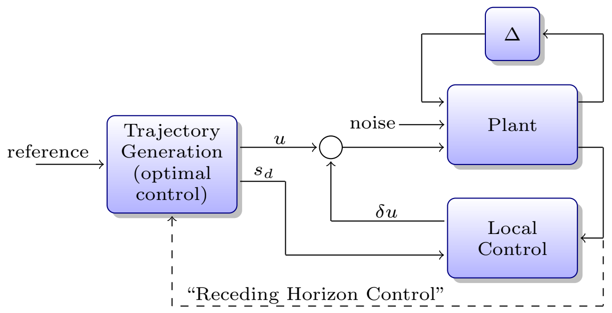
Having shown that the specification (5) can be reduced to (7), in Section 4.1 we describe a receding horizon strategy that allows the synthesis problem for a specification (7) to be reduced to a sequence of smaller problems of shorter horizon. Then, in Section 4.2, we describe its implementation, leading to the decomposition of the planner into a goal generator and a trajectory planner.

4.1 Receding Horizon Strategy

The main limitation of the synthesis of finite state automata from their LTL specifications [17] 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 10 variables, each can take any value in $\{1, \ldots, 10\}$, then there may be as many as 10^{10} nodes in the automaton. This type of computational complexity limits the application of the systhesis to relatively small problems.

Similar computational complexity is also an inherent problem in the area of constrained optimal control. Consider, for example, a trajectory generation problem as shown in Figure 1. In traditional constrained optimal control [6], trajectory generation is typically run in an open loop manner, i.e., there is no dashed arrow labeled "Receding Horizon Control" and a reference trajectory s_d is computed offline, taking into account all the possible initial conditions. An effective and well-established approach to deal with computational complexity pertaining to this problem is to "close the loop" at the trajectory generation level as shown in Figure 1 and allow control strategies to be designed and implemented 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 strategy 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).

We apply a similar idea to reduce computational complexity in the synthesis of finite state automata in order to extend the traditional receding horizon control to handle (potentially dynamic and not a priori known) environments and more general properties than stability. First, we observe that in many applications, it is not necessary to plan for the whole execution, taking into account all the possible



Δ
reference
Trajectory Generation (optimal control)
u
s_d
noise
Plant
δu
Local Control
"Receding Horizon Control"

Figure 1: A typical control system with trajectory generation implemented in a receding horizon manner. Δ models uncertainties in the plant model. The local control is implemented to account for the effect of the noise and unmodeled dynamics captured by Δ.

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, say 100 kilometers, away from the initial position of the robot. Under certain conditions, it may be sufficient to only plan out an execution for 500 meters 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 (5) 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 (5).

We consider a specification of the form (7) since, from Proposition 1, the specification (5) 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 $\Box\Diamond\psi_{g,i}$ where $i \in I_g$, 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 $\mathcal{W}_0^i \prec_{\psi_{g,i}} \mathcal{W}_j^i, \forall j \neq 0$.

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 $\mathcal{F}^i(\mathcal{W}_0^i) \preceq_{\psi_{g,i}} \mathcal{W}_0^i$ and $\mathcal{F}^i(\mathcal{W}_j^i) \prec_{\psi_{g,i}} \mathcal{W}_j^i, \forall j \neq 0$.

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 follows:

$$\begin{aligned}\Psi_j^i = &\Big((\nu \in \mathcal{W}_j^i) \ \wedge\ \Phi \ \wedge\ \Box\psi_e^e \ \wedge\ \textstyle\bigwedge_{k \in I_f} \Box\Diamond\psi_{f,k}^e\Big) \\ &\implies \Big(\textstyle\bigwedge_{k \in I_s} \Box\psi_{s,k} \ \wedge\ \Box\Diamond\big(\nu \in \mathcal{F}^i(\mathcal{W}_j^i)\big) \ \wedge\ \Box\Phi\Big),\end{aligned} \tag{8}$$

where ν is the state of the system and ψ_e^e, $\psi_{f,k}^e$ and $\psi_{s,k}$ are defined as in (7).

Essentially, 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 4 for the role of Φ in this framework.) The partial order $\mathcal{P}^i$ essentially provides a measure of "closeness" to the states satisfying $\psi_{g,i}$ (i.e., the states that constitute the progress of the system). Since each specification Ψ_j^i asserts that the system eventually reaches a state that is smaller in the partial order, it essentially 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. If the function $\mathcal{F}^i$ is chosen properly so that we essentially have to plan a short step ahead, then the automaton $\mathcal{A}_j^i$ will contain significantly less number of states than an automaton satisfying the original specification (7).

Receding Horizon Strategy: Let $I_g = \{i_1, \ldots, i_n\}$ and define a corresponding ordered set $(i_1, \ldots, i_n)$.

(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 Figure 2.

(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. (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 1 for more details.) This step corresponds to going all the way down the leftmost column in Figure 2.

(3) Switch to the automaton $\mathcal{A}_{j_2}^{i_2}$ where the index j_2 is chosen such that the current state $\nu_2 \in \mathcal{W}_{j_2}^{i_2}$. Work through the partial order $\mathcal{P}^{i_2}$ until a state $\nu_3 \in \mathcal{W}_0^{i_2}$ is reached as previously done in step (2) for the partial order $\mathcal{P}^{i_1}$. 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. (As previously noted, since for any $i \in I_g$, the union of $\mathcal{W}_1^i, \ldots, \mathcal{W}_p^i$ is the set $\mathcal{V}$ of all the states, given any $\nu_2 \in \mathcal{V}$, there exist $j_2 \in \{0, \ldots, p\}$ such that $\nu_2 \in \mathcal{W}_{j_2}^{i_2}$.) In Figure 2, 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 1. 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 (7), i.e., any execution of the system satisfies (7).

Proof. Consider an arbitrary execution σ of the system that satisfies the assumption part of (7). 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 (8). 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 point where ν_2 is reached.

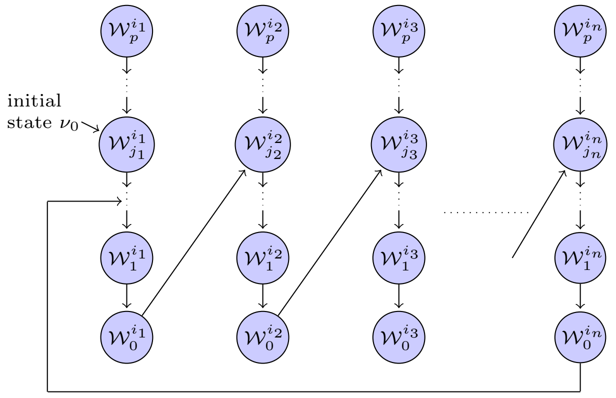


Figure 2: A graphical description of the receding horizon strategy for a special case where for each $i \in I_g$, $\mathcal{W}_j^i \prec_{\psi_{g,i}} \mathcal{W}_k^i, \forall j < k$, $F^i(\mathcal{W}_j^i) = \mathcal{W}_{j-1}^i, \forall j > 0$ and $F^i(\mathcal{W}_0^i) = \mathcal{W}_0^i$. Starting from 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.

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 p_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 p_1. According to the receding horizon strategy, the planner switches to the automaton $\mathcal{A}_k^{i_1}$ at position p_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 (8) 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 p_1 up to and including position p_2 at which the planner switches the automaton and Φ holds at position p_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, 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. $\square$

Remark 3. 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 of the plant [16]. With disturbances and modeling uncertainties, an actual execution of the system usually deviates from a reference trajectory s_d, computed by the trajectory generation component (cf. Figure 1). 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 4. The propositional formula Φ (which can be viewed as an invariant of the system) adds an assumption on the initial state of each automaton $\mathcal{A}_j^i$ and is added in order to make Ψ_j^i realizable. Without Φ, the set of initial states of $\mathcal{A}_j^i$ includes any state $\nu \in \mathcal{W}_j^i$. 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 $\Phi \equiv \mathit{True}$ 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 essentially 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 5. 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 4.2, 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, checking the realizability can be done symbolically and enumeration of states is only required when a strategy needs to be extracted [17]. Symbolic methods are known to handle large number of states, in practice, significantly better than enumeration-based methods. Hence, state explosion usually occurs at 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, our approach avoids state explosion both by considering a short-horizon problem and a specific initial state.

Remark 6. The proposed receding horizon approach is not complete. Even if there exists a control strategy that satisfies the original specification, there may not exist an invariant Φ and 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 $j \in \{0, \ldots, p\}$.

4.2 Implementation

In order to implement the receding horizon strategy described in Section 4.1, 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, essentially 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 $\Psi^i_j, i \in I_g, j \in \{0, \ldots, p\}$ as defined in (8) are realizable.

Given an invariant Φ and subsets $\mathcal{W}^i_0, \ldots, \mathcal{W}^i_p$ of $\mathcal{V}$ for each $i \in I_g$, we first construct a finite transition system $\mathbb{T}^i$ with the set of states $\{\mathcal{W}^i_0, \ldots, \mathcal{W}^i_p\}$. For each $j, k \in \{0, \ldots, p\}$, there is a transition $\mathcal{W}^i_j \to \mathcal{W}^i_k$ in $\mathbb{T}^i$ only if $j \neq k$ and the specification in (8) is realizable with $\mathcal{F}^i(\mathcal{W}^i_j) = \mathcal{W}^i_k$. This 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 $\mathcal{W}^i_j$ to $\mathcal{W}^i_0$ for all $i \in I_g$, $j \in \{1, \ldots, p\}$. (Verifying this property is essentially a graph search problem. If a path does not exist, Φ can be re-computed using a procedure described in Remark 4.) We propose an embedded control software with three components (cf. Figure 3).

Goal generator: Define 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) Evaluate whether the abstract state corresponding to the currently observed state belongs to $\mathcal{W}^{i_k}_0$. If so, 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 $\mathcal{W}^{i_k}_j$ to $\mathcal{W}^{i_k}_0$ where the index $j \in \{0, \ldots, p\}$ is chosen such that the abstract state corresponding to the currently observed state belongs to $\mathcal{W}^{i_k}_j$.

(a4) If a new path is computed in step (a3), then issue this path (i.e., a sequence $\mathcal{G} = \mathcal{W}^{i_k}_{l_0}, \ldots, \mathcal{W}^{i_k}_{l_m}$ 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 $\mathcal{W}^{i_k}_{l_\alpha} \to \mathcal{W}^{i_k}_{l_{\alpha+1}}$ 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 $\mathcal{W}^{i_k}_j$ to $\mathcal{W}^{i_k}_0$ can be efficiently solved using any graph search or shortest-path algorithm [18], 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 $(\mathcal{W}^{i_k}_{l_\alpha}, \mathcal{W}^{i_k}_{l_{\alpha'}})$ 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 $\mathcal{W}^{i_k}_{l_{\alpha'}}$.

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.

[1]This order can be defined arbitrarily. In general, its definition affects a strategy the system chooses to satisfy the specification (7) as it corresponds to the sequence of progress properties $\psi_{g,i_1}, \ldots, \psi_{g,i_n}$ the system tries to satisfy.

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

(b2) Check whether a new sequence of goal states is received from the goal generator.
If so, update $\mathcal{G}$ to this latest sequence of goal states, update q to 1, and update ν^* to null.
Otherwise, evaluate whether the abstract state corresponding to the currently observed state belongs to $\mathcal{W}^{i_k}_{l_q}$. If so, 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 (using, for example, the synthesis tool [17]) a strategy that satisfies the specification (8) 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 last received (abstract) final controlled state ς^* from the trajectory planner. In each iteration, it receives the currently observed state s of the plant. Then, it determines 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. See, for example, [11, 2, 10, 22, 19, 5], for how such a control signal can be computed.

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 4.1. Roughly speaking, the path $\mathcal{G}$ from $\mathcal{W}^i_j$ to $\mathcal{W}^i_0$ computed by the goal generator essentially defines the partial order $\mathcal{P}^i$ and the corresponding function $\mathcal{F}^i$. For a set $\mathcal{W}^i_{l_\alpha} \neq \mathcal{W}^i_0$ contained in $\mathcal{G}$, we simply let $\mathcal{W}^i_{l_{\alpha+1}} \prec \mathcal{W}^i_{l_\alpha}$ and $\mathcal{F}^i(\mathcal{W}^i_{l_\alpha}) = \mathcal{W}^i_{l_{\alpha+1}}$ where $\mathcal{W}^i_{l_{\alpha+1}}$ immediately follows $\mathcal{W}^i_{l_\alpha}$ 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 $\mathcal{W}^i_l$ to $\mathcal{W}^i_0$, it can be easily verified that the specification Ψ^i_l is realizable with $\mathcal{F}(\mathcal{W}^i_l) = \mathcal{W}^i_0$. Thus, to be consistent with the previously described receding horizon framework, we simply assign $\mathcal{W}^i_l \succ \mathcal{W}^i_0$ and $\mathcal{F}(\mathcal{W}^i_l) = \mathcal{W}^i_0$ for a set $\mathcal{W}^i_l$ not contained in $\mathcal{G}$. Note that such $\mathcal{W}^i_l$ 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 1 to conclude that the abstract plan generated by the trajectory planner ensures the correctness of the system with respect to the specification (7). 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 Figure 3. Note that since it is guaranteed to satisfy the specification (7), the desired behavior (i.e. the guarantee part of (7)) 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 5. 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. Notice the similarity of Figure 1 and Figure 3. The trajectory generation in Figure 1 is essentially decomposed into the goal generator, the trajectory planner and the continuous controller in Figure 3.

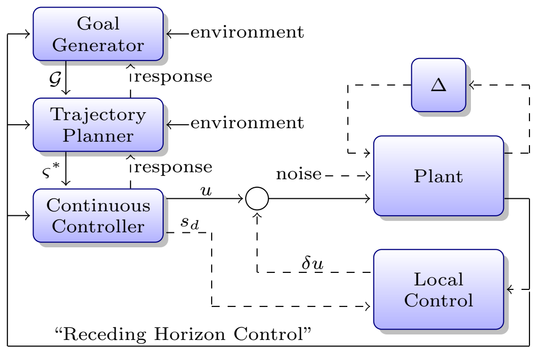


Figure 3: A system with the embedded control software implemented in a receding horizon manner. As in Figure 1, Δ models uncertainties in the plant model.

5. EXAMPLE

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 assumed to follow a fully actuated model $\dot{x}(t) = u_x(t)$ and $\dot{y}(t) = u_y(t)$ subject to the following constraints on the control effort: $u_x(t), u_y(t) \in [-1, 1], \forall t \geq 0$.

Interests in autonomous driving in urban environments were recently stimulated by the launch of the 2007 DARPA Urban Challenge [1]. In this competition, autonomous vehicles have to navigate, in a fully autonomous manner, through a partially known urban-like environment populated with (static and dynamic) obstacles and perform different tasks such as road and off-road driving, parking and visiting certain areas while obeying traffic rules. 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).

A common approach to solve the planning and control aspect of this problem is a three layer design with a mission planner computing a route (i.e., a sequence of roads to be navigated) to accomplish the given tasks, a trajectory planner computing a path (i.e., a sequence of desired positions) satisfying the traffic rules that essentially describes how the vehicle should navigate the route generated by the mission planner, and a controller computing a control signal such that the vehicle closely follows the path generated by the trajectory planner [21]. Observe how this three layer approach naturally follows our general framework for embedded control software design (cf. Figure 3) with the 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 [12], 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 4, this approach fails to handle large problems due to the state explosion issue. In this section, we show how to apply the receding horizon framework to substantially reduce computational complexity of the correct-by-construction approach.

We consider the road network shown in Figure 4 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 is defined in Figure 4. We partition the roads and intersections into $N = 282$ cells (cf. Figure 4), each of which may be occupied by an obstacle.

Given this system model, we want to design embedded control software for the vehicle based on the following desired properties and assumptions.

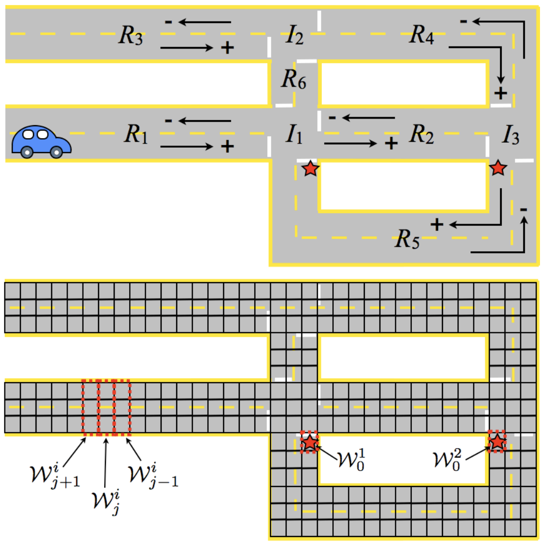


Figure 4: The road network and its partition for the autonomous vehicle example. The solid (black) lines define the states in the set $\mathcal{V}$ of the finite state model $\mathbb{D}$ used by the trajectory planner. Examples of subsets $\mathcal{W}_j^i$'s are drawn in dotted (red) rectangles. The stars indicate the areas that need to be visited infinitely often.

Desired Properties: We require that each of the two cells marked by star needs to be visited infinitely often while the following traffic rules need to be satisfied at all time.

(P1) No collision, i.e., the vehicle cannot occupied the same cell as an obstacle.

(P2) The vehicle stays in the travel lane (i.e., right lane) unless there is an obstacle blocking the lane.

(P3) The vehicle can only proceed through an intersection when the intersection is clear.

Assumptions: We assume that the vehicle starts from some point on R_1 that belongs to an obstacle-free cell and at least one of its adjacent cells is obstacle-free. In addition, the environment is assumed to satisfy the following assumptions throughout an 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.

(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.

See [22] for precise statements of properties (P1) and (P2) and assumptions (A1)–(A4) and how they can be expressed in the form of the guarantee and assumption parts of (7). Property (P3) can be expressed as a safety formula and the requirement that the vehicle visit the two cells infinitely often is essentially 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 4. First, we compute a finite state abstraction $\mathbb{D}$ of the system. Following the scheme in [22], a state ν of $\mathbb{D}$ can be written as $\nu = (\varsigma, \rho, o_1, o_2, \ldots, o_N)$ where $\varsigma \in \{1, \ldots, N\}$ 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 $i \in \{1, \ldots, N\}$, $o_i \in \{0, 1\}$ indicates whether the i^{th} cell is occupied by an obstacle. This leads to the total of $2N2^N$ possible states of $\mathbb{D}$. For any two states ν_1 and ν_2 of $\mathbb{D}$, there is a transition $\nu_1 \rightarrow \nu_2$ if the controlled state components of ν_1 and ν_2 correspond to adjacent cells.

Since the only progress property is to visit the two cells marked by star infinitely often, the set I_g in (7) 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 Figure 4. 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. Figure 4).

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 of the cells adjacent to the vehicle are obstacle-free. With this Φ, the specification (8) is realizable with $\mathcal{F}^i(\mathcal{W}_j^i) = \mathcal{W}_k^i$ for any adjacent $\mathcal{W}_j^i$ and $\mathcal{W}_k^i$ (e.g. in Figure 4, the specification (8) is realizable with both $\mathcal{F}^i(\mathcal{W}_j^i) = \mathcal{W}_{j-1}^i$ and $\mathcal{F}^i(\mathcal{W}_j^i) = \mathcal{W}_{j+1}^i$). The finite transition system $\mathbb{T}^i$ used by the goal planner can then be constructed such that there is a transition $\mathcal{W}_j^i \rightarrow \mathcal{W}_k^i$ 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[2]. Thus, the size of finite state automata synthesized by the trajectory planner to satisfy the specification (8) is completely independent of N. Using JTLV [17], 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 [15] at the continuous controller level, our approach can be potentially implemented in real-time.

A simulation result is shown in Figure 5(a), illustrating a correct execution of the vehicle 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 essentially 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 Figure 5(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 (8) 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.

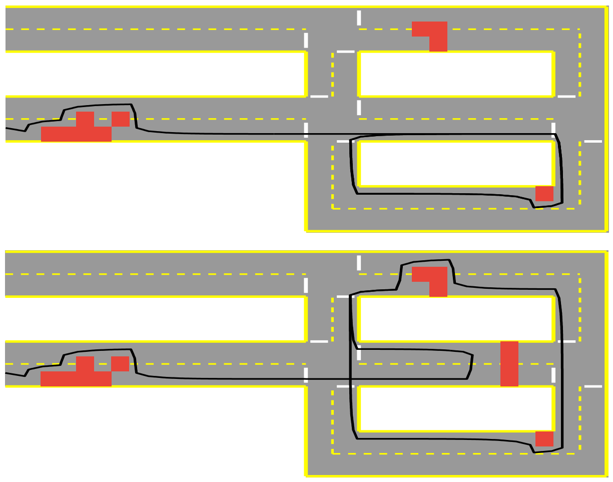


Figure 5: Simulation results with (top) no road blockage, (bottom) a road blockage on R_2.

[2]A longer horizon may be used. However, in general, the size of an automaton satisfying the specification (8) increases with the horizon length. With too short horizon, the specification (8) is generally not realizable. A good practice is to choose the shortest horizon, subject to the realizability of the resulting specification (8).

6. CONCLUSIONS AND FUTURE WORK

This paper described a receding horizon based framework that allows a computationally complex synthesis problem to be reduced to a set of significantly smaller problems. An implementation of the proposed framework was presented, leading 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. The example illustrated that the system is capable of exhibiting a correct behavior even if some of the assumptions on the environment do not hold in the actual execution.

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 mechanism has been implemented on Alice, an autonomous vehicle built at Caltech, for distributed mission and contingency management [21]. 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. Although as described in the paper, these components are to be executed sequentially, with certain assumptions on the communication channels, a distributed, asynchronous implementation of these components may still guarantee the correctness of the system.

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

7. REFERENCES

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

[2] 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*, pages 572–577, 2007.

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

[4] A. Girard, A. A. Julius, and G. J. Pappas. Approximate simulation relations for hybrid systems. *Discrete Event Dynamic Systems*, 18(2):163–179, 2008.

[5] A. Girard and G. J. Pappas. Brief paper: Hierarchical control system design using approximate simulation. *Automatica*, 45(2):566–571, 2009.

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

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

[8] A. Jadbabaie. *Nonlinear Receding Horizon Control: A Control Lyapunov Function Approach.* PhD thesis, California Institute of Technology, 2000.

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

[10] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transaction on Automatic Control*, 53(1):287–297, 2008.

[11] 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*, pages 3116–3121, April 2007.

[12] 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*, pages 766–771, 2008.

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

[14] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814(26), June 2000.

[15] 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*, pages 845–851, 2000.

[16] 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*, pages 149–174. Wiley-Interscience, 2002.

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

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

[19] P. Tabuada and G. J. Pappas. Linear time logic control of linear systems. *IEEE Transaction on Automatic Control*, 51(12):1862–1877, 2006.

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

[21] 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.

[22] 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*, Dec. 2009.