LTL receding horizon control for finite deterministic systems

Automatica 50 (2014) 399–408

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

LTL receding horizon control for finite deterministic systems☆

Xuchu Ding [a,1], Mircea Lazar [b], Calin Belta [c]

[a] Embedded Systems and Networks group, United Technologies Research Center, East Hartford, CT 06118, USA
[b] Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
[c] Department of Mechanical Engineering, Boston University, Boston, MA 02215, USA

ARTICLE INFO

Article history:
Received 7 August 2012
Received in revised form
8 October 2013
Accepted 21 October 2013
Available online 20 December 2013

Keywords:
Finite deterministic systems
Model predictive control
Formal methods
Optimal control

ABSTRACT

This paper considers receding horizon control of finite deterministic systems, which must satisfy a high level, rich specification expressed as a linear temporal logic formula. Under the assumption that time-varying rewards are associated with states of the system and these rewards can be observed in real-time, the control objective is to maximize the collected reward while satisfying the high level task specification. In order to properly react to the changing rewards, a controller synthesis framework inspired by model predictive control is proposed, where the rewards are locally optimized at each time-step over a finite horizon, and the optimal control computed for the current time-step is applied. By enforcing appropriate constraints, the infinite trajectory produced by the controller is guaranteed to satisfy the desired temporal logic formula. Simulation results demonstrate the effectiveness of the approach.

1. Introduction

This paper considers the problem of controlling a deterministic discrete-time system with a finite state-space, which is also referred to as a finite transition system. Such systems can be effectively used to capture behaviors of more complex dynamical systems, and as a result, greatly reduce the complexity of control design. A finite transition system can be constructed from a continuous system via an "abstraction" process. For example, for an autonomous robotic vehicle moving in an environment, the motion of the vehicle can be abstracted to a finite system through a partition of the environment. The set of states can be seen as a set of labels for the regions in the partition, and each transition corresponds to a controller driving the vehicle between two adjacent regions. By partitioning the environment into simplicial or rectangular regions, continuous feedback controllers that drive a robotic system from any point inside a region to a desired facet of an adjacent region have been developed for linear (Kloetzer & Belta, 2008a), multi-affine (Habets, Collins, & van Schuppen, 2006), piecewise-affine (Desai, Ostrowski, & Kumar, 1998; Habets, Kloetzer, & Belta, 2006; Wongpiromsarn, Topcu, & Murray, 2009), and non-holonomic (unicycle) (Belta, Isler, & Pappas, 2005; Lindemann, Hussein, & LaValle, 2007) dynamical models. By relating the initial continuous dynamical system and the abstract discrete finite system through simulation or bisimulation relations (Milner, 1989), the abstraction process allows one to replace the original more complex continuous system with the "equivalent" abstract system when solving a control synthesis problem.

Due to their expressivity and resemblance to natural language, temporal logics (Clarke, Peled, & Grumberg, 1999), such as linear temporal logic (LTL) and computation tree logic (CTL), have been proposed by several authors (Karaman & Frazzoli, 2009; Kloetzer & Belta, 2008a; Kress-Gazit, Fainekos, & Pappas, 2007; Loizou & Kyriakopoulos, 2004; Wongpiromsarn et al., 2009) as specification languages for control problems. In particular, LTL formulas can be used to specify persistent surveillance missions such as "pick up items at the region `pickup`, and then drop them off at the region `dropoff`, infinitely often, while always avoiding `unsafe` regions". Model checking techniques (Clarke et al., 1999) and temporal logic games (Piterman, Pnueli, & Saar, 2006) can be adapted to derive algorithms for controlling finite systems from temporal logic specifications (Karaman & Frazzoli, 2009; Kloetzer & Belta, 2008a; Kress-Gazit et al., 2007; Loizou & Kyriakopoulos, 2004; Wongpiromsarn et al., 2009).

While the works mentioned above address the temporal logic controller synthesis problem, several questions remain to be

☆ This work was partially supported by the ONR-MURI Award N00014-09-1051 at Boston University and the STW Veni Grant 10230 at Eindhoven University of Technology. The material in this paper was partially presented at the 49th IEEE Conference on Decision and Control (CDC), December 15–17, 2010 and the American Control Conference (ACC 2012), June 27–29, 2012, Montréal, Canada. This paper was recommended for publication in revised form by Associate Editor Lalo Magni under the direction of Editor Frank Allgöwer.

E-mail addresses: dingx@utrc.utc.com (X. Ding), m.lazar@tue.nl (M. Lazar), cbelta@bu.edu (C. Belta).

[1] Tel.: +1 404 452 4335; fax: +1 860 755 0019.

0005-1098/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.automatica.2013.11.030

answered. In particular, the problem of combining temporal logic controller synthesis with optimality with respect to a suitable cost function is not well understood. In Kloetzer and Belta (2008a), the authors considered a simple cost function that penalized the combined cost of the prefix and suffix of an infinite run satisfying an LTL formula. This idea was extended in Smith, Tůmová, Belta, and Rus (2011), where the goal was to minimize the maximum distance in between successful satisfactions of a given proposition, while satisfying an LTL specification.

This problem becomes even more difficult if the optimization problem depends on time-varying parameters, *e.g.,* dynamic events that occur during the operation of the plant. For traditional control problems (without temporal logic constraints) and dynamical systems, this problem can be effectively addressed using a model predictive control (MPC) paradigm (see *e.g.,* Rawlings & Mayne, 2009), which has reached a mature level in both academia and industry, with many successful implementations. The basic MPC setup consists of the following sequence of steps: at each time instant, a cost function of the current state is optimized over a finite horizon, only the first element of the optimal finite sequence of controls is applied, and the whole process is repeated at the next time instant for the new measured state. For this reason, MPC is also referred to as receding horizon control. Since the finite horizon optimization problem is solved repeatedly at each time instant, real-time dynamical events can be effectively managed. MPC has already been applied successfully to hybrid dynamical systems with mixed continuous and discrete dynamics, see, for example, Bemporad and Morari (1999), Di Cairano, Lazar, Bemporad, and Heemels (2008) and the references therein.

However, it is not yet understood how to combine a receding horizon control approach with a control strategy satisfying a temporal logic formula. The aim of this paper is to address this issue for a relevant class of systems (i.e., deterministic system with a finite state-space) and problem formulation (i.e., dynamic optimization of rewards). Specifically, the role of the receding horizon controller is to maximize over a finite horizon the accumulated rewards associated with states of the system, under the assumption that the rewards change dynamically with time and they can only be locally observed in real-time. Note that this event-triggered reward process is widely used in the coverage control literature (Li & Cassandras, 2006). The key challenge is to ensure correctness of the produced infinite trajectory and recursive feasibility of the optimization problem solved at each time step. In a constrained MPC optimization problem, which is solved recursively on-line, recursive feasibility means that the problem has a solution for all times if it has a solution for the initial state at the initial time. In this paper, we propose a control strategy that satisfies both properties for deterministic transition systems and full LTL specifications. Similar to standard MPC, where certain *terminal* constraints must be enforced in the optimization problem in order to guarantee certain properties for the system (*e.g.,* stability), correctness and recursive feasibility are also ensured via a set of suitable constraints.

This work is a combination and generalization of the results presented in Chu Ding, Belta, and Cassandras (2010) and Chu Ding, Lazar, and Belta (2012). In Ding et al. (2010), an optimization based controller was designed, which consisted of repeatedly solving a finite horizon optimal control problem every N steps and implementing the complete sequence of control actions. Its main drawback came from the inability of reacting to dynamical events (*i.e.,* rewards) triggered during the execution of the finite trajectory. In Ding et al. (2012), we removed this limitation by attaining a truly receding horizon controller for deterministic systems with finite state-spaces. In the current paper, we extend upon (Ding et al., 2012) by allowing a more general cost function, and provide full proofs and complexity analysis as they were omitted in the preliminary works (Ding et al., 2010, 2012).

This work is also related to Wongpiromsarn et al. (2009), where a provably correct control strategy was incrementally obtained by dividing the control synthesis problem into smaller subproblems in a receding horizon-like manner. The specifications were restricted to a fragment of LTL, called GR(1) (Piterman et al., 2006), which allowed for the definition of a partial order over satisfying runs. By defining a Lyapunov-like energy function that enforces the acceptance condition of an automaton, we are able to provide a complete solution for full LTL, while at the same time guaranteeing local optimality. To the best of our knowledge, this is one of the first attempts to combine temporal logic controller synthesis with local optimization techniques so that the controller reacts to environmental changes during its operation.

The remainder of the paper is organized as follows. The problem formulation and the main ingredients of the proposed approach are included in Section 2. In Section 3 we formulate the energy function that will be used in defining the terminal cost of the MPC optimization problem. The proposed receding horizon control framework and the main results are presented in Section 4. An illustrative case study is included in Section 5. Conclusions are summarized in Section 6.

2. Problem formulation and approach

In this paper, we consider a dynamical system that evolves on a finite graph by deterministically choosing an available edge at the current state. Such a system can be described by a finite deterministic transition system, which can be formally defined as follows.

Definition 2.1 (*Finite Deterministic Transition System*)**.** A finite (weighted) deterministic transition system (DTS) is a tuple $\mathcal{T} = (Q, q_0, \Delta, \omega, \Pi, h)$, where

- Q is a finite set of states;
- $q_0 \in Q$ is the initial state;
- $\Delta \subseteq Q \times Q$ is the set of transitions;
- $\omega : \Delta \to \mathbb{R}^+$ is a weight function;
- Π is a set of observations; and
- $h : Q \to 2^{\Pi}$ is the observation map.

For convenience of notation, we denote $q \to_{\mathcal{T}} q'$ if $(q, q') \in \Delta$. We assume $\mathcal{T}$ to be non-blocking, *i.e.,* for each $q \in Q$, there exists $q' \in Q$ such that $q \to_{\mathcal{T}} q'$ (such a system is also called a *Kripke structure* Browne, Clarke, & Grumberg, 1988). A *trajectory* of a DTS is an infinite sequence $\mathbf{q} = q_0 q_1 \ldots$, where $q_k \in Q$ and $q_k \to_{\mathcal{T}} q_{k+1}$ for all $k \geq 0$. A trajectory $\mathbf{q}$ generates an *output trajectory* $\mathbf{o} = o_0 o_1 \ldots$, where $o_k = h(q_k)$ for all $k \geq 0$.

Note the absence of control inputs in the definition of $\mathcal{T}$. It is assumed that a transition $(q, q') \in \Delta$ can be deterministically chosen at q. This implies that there is a one-to-one map between a trajectory $\mathbf{q} = q_0 q_1 q_2 \ldots$ and a sequence of transitions $(q_0, q_1), (q_1, q_2), \ldots$. Throughout this paper, a strategy that specifies an available transition $(q, q') \in \Delta$ at state q at time k will be referred to as a *control strategy*, or *controller*.

An example of a DTS is shown in Fig. 1. States with observation $\{\texttt{base}, \texttt{survey}, \texttt{recharge}, \texttt{unsafe}\}$ are shown in large circles with color blue, cyan, purple and black, respectively, and states with no observation are shown as small black circles.

We employ Linear Temporal Logic (LTL) for system specifications. A detailed description of the syntax and semantics of LTL is beyond the scope of this paper and can be found in Clarke et al. (1999). Roughly, an LTL formula is built up from a set of atomic propositions Π, standard Boolean operators $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), and temporal operators X (next), U (until), F (eventually), G (always). An LTL formula over Π is interpreted over an (infinite) sequence $\mathbf{o} = o_0 o_1 \ldots$, where $o_k \subseteq \Pi$

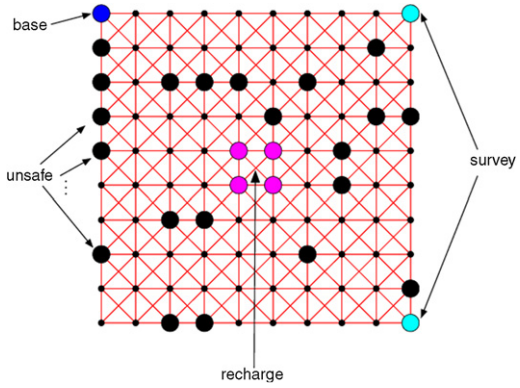


Fig. 1. An example of a finite DTS $\mathcal{T}$ (Definition 2.1) modeling the motion of a robot in a graph-like environment. In this example, $\mathcal{T}$ has 100 states, which are at the vertices of a rectangular grid with cell size 10. The weight function ω is the Euclidean distance between vertices, and there is a transition between two vertices if the Euclidean distance between them is less than 15. The set of observations is $\Pi = \{\texttt{base}, \texttt{survey}, \texttt{recharge}, \texttt{unsafe}\}$.

for all $k \geq 0$, such as the words generated by a DTS $\mathcal{T}$. A word satisfies an LTL formula ϕ if ϕ is true at the first position of the word; $\mathsf{G}\,\phi$ means that ϕ is true at all positions of the word; $\mathsf{F}\,\phi$ means that ϕ eventually becomes true in the word; $\phi_1\,\mathsf{U}\,\phi_2$ means that ϕ_1 has to hold at least until ϕ_2 is true. More expressivity can be achieved by combining the above temporal and Boolean operators (more examples will be given throughout the paper).

The goal of this paper is to synthesize trajectories $\mathbf{q}$ of $\mathcal{T} = (Q, q_0, \Delta, \omega, \Pi, h)$ satisfying a specification given as an LTL formula over Π. LTL is chosen as a desirable specification due to its rich expressivity and similarity to natural language. Examples of specifications that can be easily translated to LTL formulas include (1) *Sequence*: "first visit states satisfying a, and then states satisfying b" ($\mathsf{F}\,(a \wedge \mathsf{F}\,b)$); (2) *Coverage*: "visit states satisfying a and states satisfying b, regardless of order" ($\mathsf{F}\,a \wedge \mathsf{F}\,b$); (3) *Persistent surveillance*: "achieve a sequence task ϕ infinitely many times" ($\mathsf{G}\,\mathsf{F}\,\phi$); and (4) *Safety*: "achieve task ϕ and always avoid states satisfying c" ($\mathsf{G}\,\neg c \wedge \phi$).

The system is assumed to operate in an environment with dynamical events. In this paper, these events are modeled by a reward process $\mathcal{R} : Q \times \mathbb{N} \to \mathbb{R}^+$, *i.e.,* the reward associated with state $q \in Q$ at time k is $\mathcal{R}(q, k)$. Note that rewards are associated with states in Q in a time varying fashion. We do not make any assumptions on the dynamics governing the rewards, but we make the natural assumption that, at time k, the system can only observe the rewards in a neighborhood $\mathcal{N}(q, k) \subseteq Q$ of the current state q.

We are now ready to formulate the main problem:

Problem 2.2. Given a DTS $\mathcal{T} = (Q, q_0, \Delta, \omega, \Pi, h)$ and an LTL formula ϕ over Π, design a controller that maximizes the collected reward locally, while it ensures that the produced infinite trajectory satisfies ϕ.

Note that it does not make sense to maximize the total collected reward over an infinite trajectory. Since the rewards are time-varying and can only be observed around the current state, inspiration from the area of MPC is drawn (see, e.g. Rawlings & Mayne, 2009) with the aim of synthesizing a controller such that the rewards are maximized in a receding horizon fashion.

The main ingredients of the MPC strategy that we propose in this paper are as follows. At time k when the state is q_k, we generate a finite trajectory $q_{k+1}q_{k+2}\ldots q_{k+N}$ by solving an on-line optimization problem maximizing the collected rewards over a horizon N. The first control action (q_k, q_{k+1}) is applied, and then the optimal trajectory is computed again at time $k+1$. We consider two key properties for a receding horizon controller tailored for LTL specifications: (1) *Correctness and completeness*: the controller must generate a trajectory satisfying the given LTL formula if one exists and (2) *Recursive feasibility*: if the repeatedly solved optimization problem is feasible at initial time, then it is feasible for all iterations.

We show that a Lyapunov-like energy function defined on the product between the DTS and an automaton that is generated from the LTL formula can be used to enforce both these properties. This energy function creates a measure of "progress" towards satisfying the given formula. It will be computed off-line once, and then it will be used on-line with the receding horizon controller. We show that suitable terminal constraints placed in terms of this energy function can be used to enforce both correctness–completeness and recursive feasibility of the proposed controllers.

Remark 2.3. DTSs form a particular class of hybrid dynamical systems as considered in Bemporad and Morari (1999) and Di Cairano et al. (2008). In Bemporad and Morari (1999), convergence and recursive feasibility of the MPC strategy was guaranteed via a terminal equality constraint. In Di Cairano et al. (2008), asymptotic stability and recursive feasibility of the MPC strategy was guaranteed via a set of inequality constraints involving a hybrid control Lyapunov function. While in this paper we restrict our attention to DTSs, as opposed to general hybrid dynamical systems, the specifications that can be represented by LTL formulas are much richer and more suited for surveillance applications, compared to classical systems theory specifications, such as convergence or asymptotic stability.

3. Energy function

In this section, we first review the definition of a Büchi automaton corresponding to an LTL formula. We then describe the construction of an energy function on the states of the product between the DTS $\mathcal{T}$ and the Büchi automaton. We show how this function can be used to enforce the satisfaction of the formula.

Definition 3.1 (*Büchi Automaton*)**.** A (non-deterministic) Büchi automaton is a tuple $\mathcal{B} = (S_{\mathcal{B}}, S_{\mathcal{B}0}, \Sigma, \delta, F_{\mathcal{B}})$, where

- $S_{\mathcal{B}}$ is a finite set of states;
- $S_{\mathcal{B}0} \subseteq S_{\mathcal{B}}$ is the set of initial states;
- Σ is the input alphabet;
- $\delta : S_{\mathcal{B}} \times \Sigma \to 2^{S_{\mathcal{B}}}$ is the transition function;
- $F_{\mathcal{B}} \subseteq S$ is the set of accepting states.

We denote $s \xrightarrow{\sigma}_{\mathcal{B}} s'$ if $s' \in \delta(s, \sigma)$. An infinite sequence $\sigma_0\sigma_1\ldots$ over Σ generates trajectories $s_0s_1\ldots$ where $s_0 \in S_{\mathcal{B}0}$ and $s_k \xrightarrow{\sigma_k}_{\mathcal{B}} s_{k+1}$ for all $k \geq 0$. $\mathcal{B}$ is said to accept an infinite sequence over Σ if the sequence generates at least one trajectory of $\mathcal{B}$ that intersects the set $F_{\mathcal{B}}$ of accepting states infinitely many times.

For any LTL formula ϕ over Π, one can construct a Büchi automaton with input alphabet $\Sigma = 2^{\Pi}$ accepting all (and only) sequences over 2^{Π} that satisfy ϕ (Clarke et al., 1999). Efficient algorithms and implementations to translate an LTL formula over Π to a corresponding Büchi automaton $\mathcal{B}$ can be found in Gastin and Oddoux (2001).

Definition 3.2 (*Weighted Product Automaton*)**.** Given a weighted DTS $\mathcal{T} = (Q, q_0, \Delta, \omega, \Pi, h)$ and a Büchi automaton $\mathcal{B} = (S_{\mathcal{B}}, S_{\mathcal{B}0}, 2^{\Pi}, \delta_{\mathcal{B}}, F_{\mathcal{B}})$, their product automaton, denoted by $\mathcal{P} = \mathcal{T} \times \mathcal{B}$, is a tuple $\mathcal{P} = (S_{\mathcal{P}}, S_{\mathcal{P}0}, \Delta_{\mathcal{P}}, \omega_{\mathcal{P}}, F_{\mathcal{P}})$ where

- $S_{\mathcal{P}} = Q \times S_{\mathcal{B}}$;
- $S_{\mathcal{P}0} = \{q_0\} \times S_{\mathcal{B}0}$;

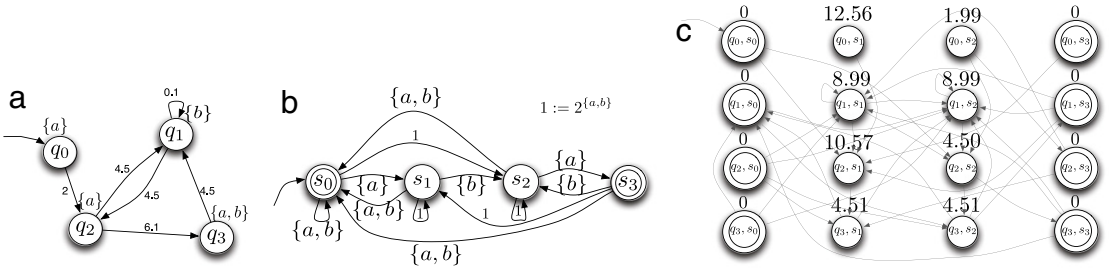


Fig. 2. The construction of product automaton and energy function. In this example, the set of observations is $\Pi = \{a, b\}$. The initial states are indicated by incoming arrows. The accepting states are marked by double-strokes. (a): A weighted DTS $\mathcal{T}$. The label atop each state indicates the set of associated observations. (*i.e.,* $\{a, b\}$ means both a and b are observed). The labels on the transitions indicate the weights. (b): Büchi automaton $\mathcal{B}$ corresponding to LTL formula $\mathsf{G}(\mathsf{F}(a \wedge \mathsf{F}\, b))$ obtained using the tool LTL2BA (Gastin & Oddoux, 2001). (c): The product automaton $\mathcal{P} = \mathcal{T} \times \mathcal{B}$ constructed according to Definition 3.2 (the weights are inherited from $\mathcal{T}$ and not shown). The number above a state $p \in S_{\mathcal{P}}$ is the energy function $V(p)$. Note that in this example, the set $F^\star_{\mathcal{P}} = F_{\mathcal{P}}$, thus $V(p)$ is the graph distance from p to any accepting states.

- $\Delta_{\mathcal{P}} \subseteq S_{\mathcal{P}} \times S_{\mathcal{P}}$ is the set of transitions, defined by: $\big((q, s), (q', s')\big) \in \Delta_{\mathcal{P}}$ iff $q \to_{\mathcal{T}} q'$ and $s \xrightarrow{h(q)}_{\mathcal{B}} s'$;
- $\omega_{\mathcal{P}} : \Delta_{\mathcal{P}} \to \mathbb{R}^+$ is the weight function defined by: $\omega_{\mathcal{P}}\big((q, s), (q', s')\big) = \omega\big((q, q')\big)$
- $F_{\mathcal{P}} = Q \times F_{\mathcal{B}}$ is the set of accepting states on $\mathcal{P}$.

We denote $(q, s) \to_{\mathcal{P}} (q', s')$ if $((q, s), (q', s')) \in \Delta_{\mathcal{P}}$. A trajectory $\mathbf{p} = (q_0, s_0)(q_1, s_1) \ldots$ of $\mathcal{P}$ is an infinite sequence such that $(q_0, s_0) \in S_{\mathcal{P}0}$ and $(q_k, s_k) \to_{\mathcal{P}} (q_{k+1}, s_{k+1})$ for all $k \geq 0$. Trajectory $\mathbf{p}$ is called accepting if and only if it intersects $F_{\mathcal{P}}$ infinitely many times.

We define the projection $\gamma_{\mathcal{T}}$ of $\mathbf{p}$ onto $\mathcal{T}$ as simply removing the automaton states, *i.e.,*

$$\gamma_{\mathcal{T}}(\mathbf{p}) = \mathbf{q} = q_0 q_1 \ldots, \quad \text{if } \mathbf{p} = (q_0, s_0)(q_1, s_1) \ldots. \tag{1}$$

We also use the projection operator $\gamma_{\mathcal{T}}$ for finite trajectories (subsequences of $\mathbf{p}$). Note that a trajectory $\mathbf{p}$ on $\mathcal{P}$ is uniquely projected to a trajectory $\gamma_{\mathcal{T}}(\mathbf{p})$ on $\mathcal{T}$. By the construction of $\mathcal{P}$ from $\mathcal{T}$ and $\mathcal{B}$, $\mathbf{p}$ is accepted if and only if $\mathbf{q} = \gamma_{\mathcal{T}}(\mathbf{p})$ satisfies the LTL formula corresponding to $\mathcal{B}$ (Clarke et al., 1999).

We now introduce a real positive function V on the states of the product automaton $\mathcal{P}$ that uses the weights $\omega_{\mathcal{P}}$ to enforce the acceptance condition of the automaton. Conceptually, this function resembles a Lyapunov or energy function. While in Lyapunov theory energy functions are used to enforce that the trajectories of a dynamical system converge to an equilibrium, the proposed "energy" function enforces that the trajectories of $\mathcal{T}$ satisfy the acceptance condition of a Büchi automaton.

Let $\mathcal{D}(p_i, p_j)$ denote the set of all finite trajectories from a state $p_i \in S_{\mathcal{P}}$ to a state $p_j \in S_{\mathcal{P}}$:

$$\mathcal{D}(p_i, p_j) = \{p_1 \ldots p_n | p_1 = p_i, p_n = p_j; \\ p_k \to_{\mathcal{P}} p_{k+1} \text{ for } k = 1, \ldots, n-1\}, \tag{2}$$

where $n \geq 2$ is an arbitrary number. Note that $\mathcal{D}(p_i, p_j)$ may not be a finite set due to possible cycles in $\mathcal{P}$. We say p_i reaches p_j, or p_j is reachable from p_i, if $\mathcal{D}(p_i, p_j) \neq \emptyset$.

Next, we define a path length function for a finite run $\mathbf{p} = p_1 \ldots p_n$:

$$L(\mathbf{p}) = \sum_{k=1}^{n-1} \omega_{\mathcal{P}}(p_k, p_{k+1}). \tag{3}$$

We can now define a distance function from a state $p \in S_{\mathcal{P}}$ to $p' \in S_{\mathcal{P}}$ as follows:

$$d(p, p') = \begin{cases} \min\limits_{\mathbf{p} \in \mathcal{D}(p,p')} L(\mathbf{p}) & \text{if } \mathcal{D}(p, p') \neq \emptyset \\ \infty & \text{if } \mathcal{D}(p, p') = \emptyset. \end{cases} \tag{4}$$

Since $\omega_{\mathcal{P}}$ is a positive-valued function, we have $d(p, p') > 0$ for all $p, p' \in S_{\mathcal{P}}$. We note that $d(p, p')$ for all $p, p' \in S_{\mathcal{P}}$ can be efficiently computed by several shortest path algorithms, such as, for example, Dijkstra's algorithm (Papadimitriou & Steiglitz, 1998).

We say that a set $A \subseteq S_{\mathcal{P}}$ is *self-reachable* if and only if all states in A can reach a state in A ($\forall p \in A, \exists p' \in A$ such that $\mathcal{D}(p, p') \neq \emptyset$). We define $F^\star_{\mathcal{P}}$ to be the *largest* self-reachable subset of $F_{\mathcal{P}}$.

Definition 3.3 (*Energy Function of a State in* $\mathcal{P}$)**.** The energy function $V(p)$, $p \in S_{\mathcal{P}}$ is defined as

$$V(p) = \begin{cases} \min\limits_{p' \in F^\star_{\mathcal{P}}} d(p, p'), & \text{if } p \notin F^\star_{\mathcal{P}} \\ 0, & \text{if } p \in F^\star_{\mathcal{P}}. \end{cases} \tag{5}$$

Clearly, $V(p) \geq 0$ for all $p \in S_{\mathcal{P}}$, $V(p) = 0$ if and only if $p \in F^\star_{\mathcal{P}}$, and $V(p) \neq \infty$ if and only if a state in the set $F^\star_{\mathcal{P}}$ is reachable from p. Thus, we note that $V(p)$ represents the minimum distance from p to the set $F^\star_{\mathcal{P}}$.

Fig. 2 shows an example of $\mathcal{T}$, $\mathcal{B}$, and their product $\mathcal{P}$, as well as the induced energy function defined on states of $\mathcal{P}$. Next we characterize some properties of V.

Theorem 3.4 (*Properties of the Energy Function*)**.** *V satisfies the following:*

(i) *If a trajectory $\mathbf{p}$ on $\mathcal{P}$ is accepting, then it cannot contain a state p where $V(p) = \infty$.*
(ii) *All accepting states in an accepting trajectory $\mathbf{p}$ are in the set $F^\star_{\mathcal{P}}$ and have energy equal to 0; all accepting states that are not in $F^\star_{\mathcal{P}}$ have energy equal to ∞.*
(iii) *For each state $p \in S_{\mathcal{P}}$, if $V(p) > 0$ and $V(p) \neq \infty$, then there exists a state p' with $p \to_{\mathcal{P}} p'$ such that $V(p') < V(p)$.*

Proof. The proof of claim (i) makes use of a contradiction argument. Suppose property (i) does not hold. Then there is an accepting state p in the trajectory $\mathbf{p}$ such that $p \notin F^\star_{\mathcal{P}}$. By definition of the acceptance condition of $\mathcal{P}$, $\mathbf{p}$ intersects $F_{\mathcal{P}}$ infinitely many times, thus there must be another accepting state $p' \in F_{\mathcal{P}}$ which is reachable from p. If $p' \in F^\star_{\mathcal{P}}$, then by the definition of $F^\star_{\mathcal{P}}$ (largest self-reachable subset of $F_{\mathcal{P}}$), p must be in $F^\star_{\mathcal{P}}$, which contradicts our assumption that $p \notin F^\star_{\mathcal{P}}$.

For the case when $p' \notin F^\star_{\mathcal{P}}$, there must be a non-trivial strongly connected component (SCC) consisting of at least one accepting state (denoted as p'') reachable from p' (Clarke et al., 1999). All states in a SCC can reach every other state in the SCC, and a SCC is trivial if it consists of a single state with no self-transition. Hence, p'' is reachable from itself. By the definition of $F^\star_{\mathcal{P}}$, $p'' \in F^\star_{\mathcal{P}}$, and consequently p' is in $F^\star_{\mathcal{P}}$, which contradicts our assumption.

(ii) follows directly from (i).

(iii) By definition of V in (5), we have $p \notin F^\star_{\mathcal{P}}$ and therefore there exists one shortest finite trajectory $p_1 p_2 \ldots p_n$ where $p_1 = p$ and $p_n \in F^\star_{\mathcal{P}}$. Bellman's Optimality Principle states that the finite run $p_2 \ldots p_n$ is the shortest run starting at p_2 to reach a state in $F^\star_{\mathcal{P}}$, and therefore $V(p) = d(p, p_2) + V(p_2)$. Since $d(p, p_2) > 0$, (iii) follows with $p' := p_2$. ■

We see that the set $F^\star_{\mathcal{P}}$ is crucial since it is the largest subset of accepting states where its member can appear in an accepting trajectory of the product automaton, and $V(p)$ is the "distance" from state p to this set of states. We refer to the value of $V(p)$ at state p as the "energy of the state". From Theorem 3.4, we see that satisfying the LTL formula is equivalent to reaching states where $V(p) = 0$ for infinitely many times.

We propose Algorithm 1 to obtain the set $F^\star_{\mathcal{P}}$ and the energy function V. This algorithm obtains the largest self-reachable subset of $F_{\mathcal{P}}$ by construction, because it starts with the whole set $F_{\mathcal{P}}$ and prunes out one by one states that cannot reach a state in itself, until all states in the set satisfy the definition of a self-reachable set.

Algorithm 1 Algorithm to compute $V(p)$, given a product automaton $\mathcal{P} = (S_{\mathcal{P}}, S_{\mathcal{P}0}, \delta_{\mathcal{P}}, \omega_{\mathcal{P}}, F_{\mathcal{P}})$, for all $p \in S_{\mathcal{P}}$

1: Compute $d(p, p')$ for all $p \in S_{\mathcal{P}}$ and $p' \in F_{\mathcal{P}}$.
2: Set $F^\star_{\mathcal{P}} = F_{\mathcal{P}}$.
3: **while** there exist $q \in F^\star_{\mathcal{P}}$ such that

$$\min_{p \in F^\star_{\mathcal{P}}} d(q, p) = \infty$$

do
4: Remove q from $F^\star_{\mathcal{P}}$.
5: **end while**
6: Obtain $V(p)$ using definition (5) for all $p \in S_{\mathcal{P}}$.

4. Design of receding horizon controllers

In this section, we present a solution to Problem 2.2. The central component of our control design is a state-feedback controller operating on the product automaton that optimizes finite trajectories over a pre-determined, fixed horizon N, subject to certain constraints. These constraints ensure that the energy of states on the product automaton decreases in finite time, thus guaranteeing that progress is made towards the satisfaction of the LTL formula. Note that the proposed controller does not enforce the energy to decrease at each time-step, but rather that it eventually decreases. The finite trajectory returned by the receding horizon controller is projected onto $\mathcal{T}$, the controller applies the first transition, and this process is repeated again at the next time-step.

In this section, we first describe the receding horizon controller and show that it is feasible (a solution exists) at all time-steps $k \in \mathbb{N}$. Then, we present the general control algorithm and show that it always produces (infinite) trajectories satisfying the given LTL formula.

4.1. Receding horizon controller

In order to explain the working principle of the controller, we first define a finite *predicted trajectory* on $\mathcal{P}$ at time k. Denote the current state at time k as p_k. A predicted trajectory of horizon N at time k is a finite sequence $\mathbf{p}_k := p_{1|k} \dots p_{N|k}$, where $p_{i|k} \in S_{\mathcal{P}}$ for all $i = 1, \dots, N$, $p_{i|k} \to_{\mathcal{P}} p_{i+1|k}$ for all $i = 1, \dots, N-1$ and $p_k \to_{\mathcal{P}} p_{1|k}$. Here, $p_{i|k}$ is a notation used frequently in MPC, which denotes the ith state of the predicted trajectory at time k. We denote the set $\mathbf{P}(p_k, N)$ as the set of all finite trajectories of horizon N from a state $p_k \in S_{\mathcal{P}}$. Note that the finite predicted trajectory $\mathbf{p}_k$ of $\mathcal{P}$ uniquely projects to a finite trajectory $\mathbf{q}_k := \gamma_{\mathcal{T}}(\mathbf{p}_k)$ of $\mathcal{T}$.

For the current state q_k at time k, we denote the observed reward at any state $q \in Q$ as $R_k(q)$, and we have that

$$R_k(q) = \begin{cases} \mathcal{R}(q, k) & \text{if } q \in \mathcal{N}(q_k, k) \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Note that $\mathcal{R}(q, k) = 0$ if $q \notin \mathcal{N}(q_k, k)$ because the rewards outside of the neighborhood cannot be observed. We can now define the *predicted reward* associated with a predicted trajectory $\mathbf{p}_k \in \mathbf{P}(p_k, N)$ at time k. The predicted reward of $\mathbf{p}_k$, denoted as $\Re_k(\mathbf{p}_k)$, is simply the amount of accumulated rewards by $\gamma_{\mathcal{T}}(\mathbf{p}_k)$ of $\mathcal{T}$:

$$\Re_k(\mathbf{p}_k) = \sum_{i=1}^{N} R_k\left(\gamma_{\mathcal{T}}(p_{i|k})\right). \tag{7}$$

The receding horizon controller executed at the initial state at time $k = 0$ is described next. This is a special case because the initial state of $\mathcal{P}$ is not unique, and as a result we can pick any initial state of $\mathcal{P}$ from the set $S_{\mathcal{P}0} = \{q_0\} \times S_{\mathcal{B}0}$. We denote the controller executed at the initial state as $\mathtt{RH}^0(S_{\mathcal{P}0})$, and we define it as follows

$$\begin{aligned} \mathbf{p}^\star_0 &= \mathtt{RH}^0(S_{\mathcal{P}0}) \\ &:= \operatorname*{arg\,max}_{\mathbf{p}_0 \in \{\mathbf{P}(p_0,N) \mid V(p_0) < \infty\}} \Re_0(\mathbf{p}_0). \end{aligned} \tag{8}$$

The controller maximizes the predicted cumulative rewards over all possible projected trajectories over horizon N initiated from a state $p_0 \in S_{\mathcal{P}0}$ where the energy is finite, and returns the optimal projected trajectory $\mathbf{p}^\star_0$. The requirement that $V(p_0) < \infty$ is critical because otherwise, the trajectory starting from p_0 cannot be accepting. If there does not exist p_0 such that $V(p_0) < \infty$, then an accepting trajectory does not exist and there is no trajectory of $\mathcal{T}$ satisfying the LTL formula (*i.e.,* Problem 2.2 has no solution).

Lemma 4.1 (*Feasibility of (8)*)**.** *Optimization problem (8) always has at least one solution if there exists p_0 such that $V(p_0) < \infty$.*

Proof. The proof follows from the fact that $\mathcal{T}$ is non-blocking, and thus the set $\mathbf{P}(p_0, N)$ is not empty. ■

Next, we present the receding horizon control algorithm for any time instant $k = 1, 2, \dots$ and corresponding state $p_k \in S_{\mathcal{P}}$. This controller is of the form

$$\mathbf{p}^\star_k = \mathtt{RH}(p_k, \mathbf{p}^\star_{k-1}) \tag{9}$$

i.e., it depends both on the current state p_k and the optimal predicted trajectory $\mathbf{p}^\star_{k-1} = p^\star_{1|k-1} \dots p^\star_{N|k-1}$ obtained at the previous time-step. Note that, by the nature of a receding horizon control scheme, the first control of the previous predicted trajectory is always applied. Therefore, we have the following equality

$$p_k = p^\star_{1|k-1}, \quad k = 1, 2, \dots. \tag{10}$$

As it will become clear in the text below, $\mathbf{p}^\star_{k-1}$ is used to enforce repeated executions of this controller to eventually reduce the energy of the state on $\mathcal{P}$ to 0.

We define controller (9) with the following three cases:

4.1.1. Case 1. $V(p_k) > 0$ and $V(p^\star_{i|k-1}) \neq 0$ for all $i = 1, \dots, N$

In this case, the receding horizon controller is defined as follows.

$$\begin{aligned} \mathbf{p}^\star_k &= \mathtt{RH}(p_k, \mathbf{p}^\star_{k-1}) \\ &:= \operatorname*{arg\,max}_{\mathbf{p}_k \in \mathbf{P}(p_k,N)} \Re_k(\mathbf{p}_k), \\ &\quad \text{subject to: } V(p_{N|k}) < V(p^\star_{N|k-1}). \end{aligned} \tag{11}$$

The key to guarantee that the energy of the states on $\mathcal{P}$ eventually decreases is the terminal constraint $V(p_{N|k}) < V(p^\star_{N|k-1})$, *i.e.,* the optimal finite predicted trajectory $p^\star_k$ must end at a state with lower energy than that of the previous predicted trajectory $p^\star_{k-1}$. This terminal constraint mechanism is graphically illustrated in Fig. 3.

To verify the feasibility of the optimization problem under this constraint, we make use of the third property of V in Theorem 3.4. Namely, each state with positive finite energy can make a transition to a state with strictly lower energy.

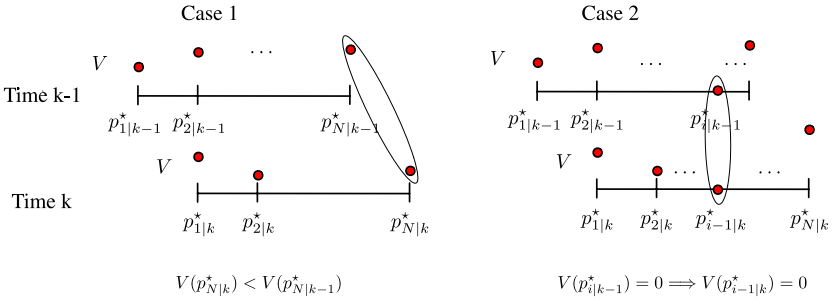


Fig. 3. Constraints enforced for the receding horizon control law $\mathbf{p}^\star_k = \mathtt{RH}(p_k, \mathbf{p}^\star_{k-1})$ for Cases 1 and 2.

Lemma 4.2 (*Feasibility of (11)*)**.** *Optimization problem (11) always has at least one solution if $V(p_k) < \infty$.*

Proof. Given $\mathbf{p}^\star_{k-1} = p^\star_{1|k-1} \dots p^\star_{N|k-1}$, since $p_k = p^\star_{1|k-1}$, we have $p_k \to_{\mathcal{P}} p^\star_{2|k-1}$. Therefore, we can construct a finite predicted trajectory $\mathbf{p}_k = p_{1|k} \dots p_{N|k}$ where $p_{i|k} = p^\star_{i+1|k-1}$ for all $i = 1, \dots, N-1$. Using Theorem 3.4(iii), there exists a state p where $p_{N-1|k} \to_{\mathcal{P}} p$ such that $V(p) < V(p_{N-1|k})$. Setting $p_{N|k} = p$, the finite trajectory $\mathbf{p}_k = p_{1|k} \dots p_{N|k} \in \mathbf{P}(p_k, N)$ satisfies the constraint $V(p_{N|k}) < V(p^\star_{N|k-1})$, and therefore (11) has at least one solution. ■

4.1.2. Case 2. $V(p_k) > 0$ and there exists $i \in \{1, \dots, N\}$ with $V(p^\star_{i|k-1}) = 0$

We denote $i^0(\mathbf{p}^\star_{k-1})$ as the index of the first occurrence in $\mathbf{p}^\star_{k-1}$ where the energy is 0, *i.e.*, $V(p^\star_{i^0(\mathbf{p}^\star_{k-1})|k-1}) = 0$. We then propose the following controller.

$$\begin{aligned}\mathbf{p}^\star_k &= \mathtt{RH}(p_k, \mathbf{p}^\star_{k-1})\\ &:= \underset{\mathbf{p}_k \in \mathbf{P}(p_k,N)}{\arg\max}\ \Re_k(\mathbf{p}_k),\\ &\quad \text{subject to: } V(p^\star_{i^0(\mathbf{p}^\star_{k-1})-1|k}) = 0. \end{aligned} \tag{12}$$

Namely, this controller enforces a state in the optimal predicted trajectory to have 0 energy if the previous predicted trajectory contains such a state. This constraint is illustrated in Fig. 3. Note that, if $i^0(\mathbf{p}^\star_{k-1}) = 1$, then from (10), the current state p_k is such that $V(p_k) = 0$, and Case 2 does not apply but Case 3 (described below) applies instead.

Lemma 4.3 (*Feasibility of (12)*)**.** *Optimization problem (12) always has at least one solution if $V(p_k) < \infty$.*

Proof. Given $\mathbf{p}^\star_{k-1} = p^\star_{1|k-1} \dots p^\star_{N|k-1}$, since $p_k = p^\star_{1|k-1}$, we have $p_k \to_{\mathcal{P}} p^\star_{2|k-1}$. Therefore, we can construct a finite predicted trajectory $\mathbf{p}_k = p_{1|k} \dots p_{N|k}$ where $p_{i|k} = p^\star_{i+1|k-1}$ for all $i = 1, \dots, N-1$. If we let $p_{N|k}$ to be any state where $p_{N-1|k} \to_{\mathcal{P}} p_{N|k}$ and $V(p_{N|k}) < \infty$, then $\mathbf{p}_k = p_{1|k} \dots p_{N|k} \in \mathbf{P}(p_k, N)$ satisfies the constraint. Theorem 3.4(iii) guarantees that such a state $p_{N|k}$ exists. ■

4.1.3. Case 3, $V(p_k) = 0$

In this case, the terminal constraint is that the energy value of the terminal state is finite. The controller is defined as follows.

$$\begin{aligned}\mathbf{p}^\star_k &= \mathtt{RH}(p_k, \mathbf{p}^\star_{k-1})\\ &:= \underset{\mathbf{p}_k \in \mathbf{P}(p_k,N)}{\arg\max}\ \Re_k(\mathbf{p}_k)\\ &\quad \text{subject to: } V(p_{N|k}) < \infty. \end{aligned} \tag{13}$$

Lemma 4.4 (*Feasibility of (13)*)**.** *Optimization problem (13) always has at least one solution.*

Proof. If $V(p_k) = 0$, then there exists $p_{1|k}$ such that $p_k \to_{\mathcal{P}} p_{1|k}$ and $V(p_{1|k}) < \infty$ (if not, then $V(p_k)$ must equal to ∞). From Theorem 3.4(iii), we have that there exists $p_{2|k}$ such that $p_{1|k} \to_{\mathcal{P}} p_{2|k}$ and $V(p_{2|k}) < V(p_{1|k}) < \infty$. By induction, there exists $\mathbf{p}_k \in \mathbf{P}(p_k, N)$ such that $V(p_{N|k}) < \infty$. ■

Remark 4.5. The proposed receding horizon control law is designed using an extension of the terminal constraint approach in model predictive control (Rawlings & Mayne, 2009) to finite deterministic systems. The particular setting of the Büchi acceptance condition, combined with the energy function V, makes it possible to obtain a non-conservative analogy of the terminal constraint approach, via either a terminal inequality condition (11) or a terminal equality condition (12).

4.2. Control algorithm and its correctness

The overall control strategy for the transition system $\mathcal{T}$ is given in Algorithm 2. After the off-line computation of the product automaton and the energy function, the algorithm applies the receding horizon controller $\mathtt{RH}^0(S_{\mathcal{P}0})$ at time $k = 0$, or $\mathtt{RH}(p_k, \mathbf{p}^\star_{k-1})$ at time $k > 0$. At each iteration of the algorithm, the receding horizon controller returns the optimal predicted trajectory $\mathbf{p}^\star_k$. The immediate transition $(p_k, p^\star_{1|k})$ is applied on $\mathcal{P}$ and the corresponding transition $(q_k, \gamma_{\mathcal{T}}(p^\star_{1|k}))$ is applied on $\mathcal{T}$. This process is then repeated at time $k+1$.

First, we show that the receding horizon controllers used in Algorithm 2 are always feasible. We use a recursive argument, which shows that if the problem is feasible for the initial state, or at time $k = 0$, then it remains feasible for all future time-steps $k = 1, 2, \dots$.

Theorem 4.6 (*Recursive Feasibility*)**.** *If there exists $p_0 \in S_{\mathcal{P}0}$ such that $V(p_0) \neq \infty$, then $\mathtt{RH}^0(S_{\mathcal{P}0})$ is feasible and $\mathtt{RH}(p_k, \mathbf{p}^\star_{k-1})$ is feasible for all $k = 1, 2, \dots$.*

Proof. From Lemma 4.1, $\mathtt{RH}^0(S_{\mathcal{P}0})$ is feasible. From the definition of $V(p)$, for all $p \in S_{\mathcal{P}}$, if $p \to_{\mathcal{P}} p'$, then $V(p') < \infty$ if and only if $V(p) < \infty$. Since $\mathtt{RH}^0(S_{\mathcal{P}0})$ is feasible, we have $p_1 = p^\star_{1|0}$ and thus $V(p_1) < \infty$. At each time $k > 0$, if $V(p_k) < \infty$, from Lemmas 4.2–4.4, we have that controller $\mathtt{RH}(p_k, \mathbf{p}^\star_{k-1})$ is feasible. Since $p_{k+1} = p^\star_{1|k}$, we have $V(p_{k+1}) < \infty$. Using induction we have that $\mathtt{RH}(p_k, \mathbf{p}^\star_{k-1})$ is feasible for all $k = 1, 2, \dots$. ■

Finally, we show that Algorithm 2 always produces an infinite trajectory satisfying the given LTL formula ϕ, giving a solution to Problem 2.2.

Theorem 4.7 (*Correctness of Algorithm 2*)**.** *Assume that there exists a satisfying run originating from q_0 for a transition system $\mathcal{T}$ and an LTL formula ϕ. Then, Algorithm 2 produces an (infinite) trajectory $\mathbf{q} = q_0 q_1 \dots$ satisfying ϕ.*

Algorithm 2 Receding horizon control algorithm for $\mathcal{T} = (Q, q_0, \Delta, \omega, \Pi, h)$, given an LTL formula ϕ over Π

Executed Off-line:

1: Construct a Büchi automaton $\mathcal{B} = (S_{\mathcal{B}}, S_{\mathcal{B}0}, 2^{\Pi}, \delta_{\mathcal{B}}, F_{\mathcal{B}})$ corresponding to ϕ.
2: Construct the product automaton $\mathcal{P} = \mathcal{T} \times \mathcal{B} = (S_{\mathcal{P}}, S_{\mathcal{P}0}, \Delta_{\mathcal{P}}, \omega_{\mathcal{P}}, F_{\mathcal{P}})$. Find $V(p)$ for all $p \in S_{\mathcal{P}}$.

Executed On-line:

3: **if** there exists $p_0 \in S_{\mathcal{P}0}$ such that $V(p_0) \neq \infty$ **then**
4: Set $k = 0$.
5: Observe rewards for all $q \in \mathcal{N}(q_0, k)$ and obtain $R_0(q)$.
6: Obtain $\mathbf{p}^\star_0 = \mathtt{RH}^0(S_{\mathcal{P}0})$.
7: Implement transition $(p_0, p^\star_{1|0})$ on $\mathcal{P}$ and transition $(q_0, \gamma_{\mathcal{T}}(p^\star_{1|0}))$ on $\mathcal{T}$.
8: Set $k = 1$
9: **loop**
10: Observe rewards for all $q \in \mathcal{N}(q_k, k)$ and obtain $R_k(q)$.
11: Obtain $\mathbf{p}^\star_k = \mathtt{RH}(p_k, \mathbf{p}^\star_{k-1})$.
12: Implement transition $(p_k, p^\star_{1|k})$ on $\mathcal{P}$ and transition $(q_k, \gamma_{\mathcal{T}}(p^\star_{1|k}))$ on $\mathcal{T}$.
13: Set $k \leftarrow k + 1$
14: **end loop**
15: **else**
16: There is no run originating from q_0 that satisfies ϕ.
17: **end if**

Proof. If there exists a satisfying run originating from q_0, then there exists a state $p_0 \in S_{\mathcal{P}0}$ such that $V(p_0) < \infty$. Therefore, from Theorem 4.6, the receding horizon controller is feasible for all $k > 0$, and Algorithm 2 will always produce an infinite trajectory $\mathbf{q}$.

At each state p_k at time $k > 0$, if $V(p_k) > 0$, then either Case 1 or Case 2 of the controller $\mathtt{RH}(p_k)$ applies. If Case 1 applies, since $V(p^\star_{k|N}) > V(p^\star_{k+1|N}) > V(p^\star_{k+2|N}) \ldots$, there exists $j > k$ such that $V(p^\star_{j|N}) = 0$. This is because the state-space $S_{\mathcal{P}}$ is finite, and therefore, there is only a finite number of possible values for the energy function $V(p)$. At time j, Case 2 of the proposed controller becomes active until time $l = j + i^0(p^\star_j)$, where $V(p_l) = 0$. Therefore, for each time k, if $V(p_k) > 0$, there exists $l > k$ such that $V(p_l) = 0$ by repeatedly applying the receding horizon controller. If $V(p_k) = 0$, then Case 3 of the proposed controller applies, in which case either $V(p_{k+1}) = 0$ or $V(p_{k+1}) > 0$. In either case, using the previous argument, there exists $j > k$ where $V(p_j) = 0$.

Therefore, at any time k, there exists $j > k$ where $V(p_j) = 0$. Furthermore, since j is finite, we can conclude that the number of times where $V(p_k) = 0$ is infinite. By the definition of $V(p)$, $p \in S_{\mathcal{P}}$, $V_k = 0$ is equivalent to $p_k \in F^\star_{\mathcal{P}} \subseteq F_{\mathcal{P}}$. Therefore, the trajectory $\mathbf{p}$ is accepting. The trajectory produced on $\mathcal{T}$ is exactly the projection $\mathbf{q} = \gamma_{\mathcal{T}}(\mathbf{p})$, and thus, it can be concluded that $\mathbf{q}$ satisfies ϕ, which completes the proof. ■

Remark 4.8. In this paper, we focus on a cost function in the form of (7). However, more general cost functions can be easily accommodated. For example, we can define a cost associated with state q at time k as $C_k(q)$ (in addition to the rewards $R_k(q)$), and the following combined cost function:

$$\sum_{i=1}^{N} R_k\left(\gamma_{\mathcal{T}}(p_{i|k})\right) - C_k\left(\gamma_{\mathcal{T}}(p_{i|k})\right).$$

The main properties of our proposed receding horizon control algorithm, namely recursive feasibility and eventual correctness of the output trajectory, still hold when using this cost function.

4.3. Complexity

The complexity of the off-line portion of Algorithm 2 depends on the size of $\mathcal{P}$. Denoting $|\phi|$ as the length of a formula ϕ (which is the total number of all symbols and operators), from Gastin and Oddoux (2001), a Büchi automaton translated from an LTL formula contains at most $|\phi| \times 2^{|\phi|}$ states.[2] Therefore, denoting $|S|$ as the cardinality of a set S, the size of $S_{\mathcal{P}}$ is bounded by $|Q| \times |\phi| \times 2^{|\phi|}$.

Remark 4.9. The problem of the exponential blow-up caused by the construction of the automaton accepting the language satisfying a LTL formula is well known (Clarke et al., 1999). Recently, there have been advances in controller synthesis for fragments of LTL such as the General Reactivity fragment (GR(1)), that is able to express almost all properties of practical interest, but for which automata synthesis is polynomial (Piterman et al., 2006). Since our method does not depend on how the Büchi Automaton is constructed, we can take advantage of any such work in reducing the size of the automaton.

The computation in Algorithm 1 involves the computation of $d(p_i, p_j)$ for all $p_j \in F_{\mathcal{P}}$ and checking the termination condition for the WHILE loop. The first task requires $|F_{\mathcal{P}}|$ runs of Dijkstra's algorithm. Each run of Dijkstra's algorithm is linear in the size of the product automaton. For the second, the WHILE loop requires at most $|F_{\mathcal{P}}|^3$ checks to see if $d(p_i, p_j)$ (already computed) is ∞ or not. The last step in Algorithm 1 requires at most $|F_{\mathcal{P}}|^2$ numerical comparisons. Overall, the complexity of Algorithm 1 is $O(|F_{\mathcal{P}}|^3 + |F_{\mathcal{P}}|^2 + |S_{\mathcal{P}}|^2 \times |F_{\mathcal{P}}|)$. Note that this algorithm is run only once off-line.

The complexity of the on-line portion of Algorithm 2 is highly dependent on the horizon N. If the maximal number of transitions at each state of $\mathcal{P}$ is $\Delta^{\max}_{\mathcal{P}}$, then the complexity at each iteration of the receding horizon controller is bounded by $(\Delta^{\max}_{\mathcal{P}})^N$, assuming a depth first search algorithm is used to find the optimal trajectory. It may be possible to reduce this complexity from exponential to polynomial if one applies a more efficient graph search algorithm using Dynamic Programming. We will explore this direction in future research.

4.4. Extensions

Even though our approach assumes that the underlying system $\mathcal{T}$ is deterministic, the results presented here can be easily extended to non-deterministic systems. The simpler extension is for the case when the LTL formula can be translated to a deterministic Büchi automaton. The procedure would start with the construction of the product automaton $\mathcal{P} = \mathcal{T} \times \mathcal{B}$ using Definition 3.2, with the difference that the weights on the transitions of $\mathcal{T}$ and $\mathcal{P}$ would not be simply defined as given in this paper, but rather as a cost associated with each non-deterministic action. In this case, we would then define the set $A \subseteq S_{\mathcal{P}}$ as *self-reachable* if it can reach another state in A, and the energy at each state of $\mathcal{P}$ as the number of steps required to reach the set A. At each time step, the controller would then be required to solve a Büchi game, similar to the control strategy used in Kloetzer and Belta (2008b). Similar to Kloetzer and Belta (2008b), this method would be restricted to LTL formulae that can be translated to deterministic Büchi automata, because the product between two non-deterministic systems is not well-defined.

Removing this restriction is not straightforward. Possible future directions to address this limitation point to game-theoretical approaches, such as Henzinger and Piterman (2006). Another possible direction is to translate the specification to a deterministic Rabin Automata (DRA) (Baier, Katoen, & Larsen, 2008). However, such methods would require the added computational complexity to deal with a DRA as it can be much larger than the equivalent Büchi automaton. In this paper, we decided to only focus on deterministic transition system to keep the notation to a minimum.

[2] In practice, this upper limit is almost never reached (see Kloetzer & Belta, 2008a).

5. Software implementation and case studies

The control framework presented in this paper was implemented as a user friendly software package, available at http://hyness.bu.edu/LTL_MPC.html. The tool takes as input the finite transition system $\mathcal{T}$, an LTL formula ϕ, the horizon N, and a function $\mathcal{R}(q, k)$ that generates the time-varying rewards defined on the states of $\mathcal{T}$. It executes the control algorithm outlined in Algorithm 2, and produces a trajectory in $\mathcal{T}$ that satisfies ϕ and maximizes the rewards collected locally with the proposed receding horizon control laws. This tool uses the LTL2BA (Gastin & Oddoux, 2001) tool for the translation of an LTL formula to a Büchi automaton. We note that very recently LTL2BA was updated and improved in Babiak, Křetínskỳ, Řehák, and Strejček (2012), and the improved version can be used instead.

We illustrate the use of the tool on the example defined in Fig. 1. We consider the following LTL formula, which expresses a robotic surveillance task:

$$\begin{aligned}\phi := {} & \mathsf{G}\,\mathsf{F}\,\texttt{base}\\ & \wedge \mathsf{G}\,(\texttt{base} \longrightarrow \mathsf{X}\,\neg\texttt{base}\,\mathsf{U}\,\texttt{survey})\\ & \wedge \mathsf{G}\,(\texttt{survey} \longrightarrow \mathsf{X}\,\neg\texttt{survey}\,\mathsf{U}\,\texttt{recharge})\\ & \wedge \mathsf{G}\,\neg\texttt{unsafe}. \end{aligned} \tag{14}$$

The first line of ϕ, $\mathsf{G}\,\mathsf{F}\,\texttt{base}$, enforces that the state with observation `base` is repeatedly visited (possibly for uploading data). The second line ensures that after `base` is reached, the system is driven to a state with observation `survey`, before going back to `base`. Similarly, the third line ensures that after reaching `survey`, the system is driven to a state with observation `recharge`, before going back to `survey`. The last line ensures that, for all times, the states with observation `unsafe` are avoided.

We assume that, at each state $q \in Q$, the rewards at state q' can be observed if the Euclidean distance between q and q' is less than or equal to 25. In the first case study, we define $\mathcal{R}(q, k)$ as follows. At time $k = 0$, there is a 50% chance that a reward value $\mathcal{R}(q, 0)$ is associated to state q, and the actual value is generated randomly from a uniform distribution in the range [10, 25]. Similarly, at each subsequent time $k > 0$, the reward is reassigned randomly from a uniform distribution in the range of [10, 25]. In this case study, the states with rewards can be seen as "targets", and the reward values can be seen as the "amount of interest" associated with each target. The control objective of maximizing the collected rewards can be interpreted as maximizing the information gathered from surveying states with high interest. We note that, in this case study, the rewards function varies with the highest possible speed (it can change at each time-step).

By applying the method described in the paper, our software package first translates ϕ to a Büchi automaton $\mathcal{B}$, which has 12 states. This procedure took 0.5 s on a Macbook Pro with a 2.2 GHz Quad-core CPU. Since $\mathcal{T}$ contains 100 states, we have $|S_{\mathcal{P}}| = 1200$. The generation of the product automaton $\mathcal{P}$ and the computation of the energy function V took 4 s. In this case study, we chose the horizon $N = 4$. By applying Algorithm 2, the first four snapshots of the system trajectory are shown in Fig. 4. Each iteration of Algorithm 2 took around 1–3 s.

We applied the control algorithm for 100 time steps and plotted the results in Fig. 5. At the top, we plot the energy $V(p)$ at the each time-step. We see that after 48 time-steps, the energy is 0, meaning that an accepting state is reached. Note that each time an accepting state is reached, the system visits the `base`, `survey` and `recharge` states at least once *i.e.,* one cycle of the surveillance mission task (`base – survey – recharge`) is completed. An example video of the evolution of the system trajectory is also available at http://hyness.bu.edu/LTL_MPC.html.

Since we have chosen a fast varying rewards function (changes at each time-step) $\mathcal{R}(q, k)$, it can be seen from Fig. 5 that the plot of the energy function is almost always decreasing. The controller often chooses to satisfy the terminal constraints (for recursive feasibility and correctness guarantees), instead of choosing to increase the energy function so that a reward can be collected. Therefore, in this case, the constraint on the energy function dominates the maximization of the local rewards. However, we note that in practical applications where receding horizon controllers are applied, the dynamics governing the time varying costs or rewards function is typically slower than the speed of the controller.

For comparison, we generated two additional case studies, with the same settings as above, except that the rewards function $\mathcal{R}(q, k)$ varies more slowly (we used the same sequence of random numbers in order to produce comparable results). The plots of the energy function versus iterations are shown in Fig. 6. It can be seen that the controller now chooses to increase the energy more often, and in most cases it takes more iterations to complete a `base – survey – recharge` cycle. This also poses an interesting trade-off between the performance in terms of the collection of rewards and the speed of finishing each iteration of the mission objective.

5.1. Performance analysis

In this section, we present a brief study of the controller performance in terms of rewards accumulation, as a function of the horizon length N. Note that the receding horizon must enforce the terminal constraints as listed in (11)–(13), therefore the control strategy is sometimes "greedy", *i.e.,* the enforcement of the terminal constraints may take precedence over the collection of local rewards.

N	1	3	4
Average rewards per iteration	0.22	1.19	1.67
Average time per iteration (s)	0.002	0.12	1.07

In the table above, we compared the results for the cases where the horizon length N is chosen to be 1, 3 and then 4 respectively. The other settings are kept the same as the previous case study in which the rewards function varies at every 3 time steps. To make a fair comparison, we used the same time-varying reward function $\mathcal{R}(q, k)$. We see that the performance (*i.e.,* the collected rewards) of the controller is directly affected by the horizon length. In short, the performance of the controller is improved by increasing N. However, increasing N will also increase the computation time required at each time-step. Therefore, these results demonstrate a trade-off that must be made between the controller performance and the computation time required in each iteration.

An interesting case is when $N = 1$. In this case, the controller will always only choose to satisfy the terminal constraints instead of collecting rewards (the rewards collected in this case are only incidental). The proposed receding horizon controller is still useful for producing a provably correct trajectory satisfying the given LTL formula, but the controller does not spend any additional control effort on gaining rewards.

6. Conclusion and final remarks

In this paper, a receding horizon control framework that optimizes the trajectory of a finite deterministic system locally, while guaranteeing that the infinite trajectory satisfies a given linear temporal logic formula, was proposed. The optimization criterion was defined as maximization of time-varying rewards associated with the states of the system. A provably-correct control strategy based on the definition of an energy-like function enforcing the acceptance condition of an automaton was developed. The proposed framework brings together ideas and techniques from model predictive control and formal synthesis, and we believe it benefits both areas.

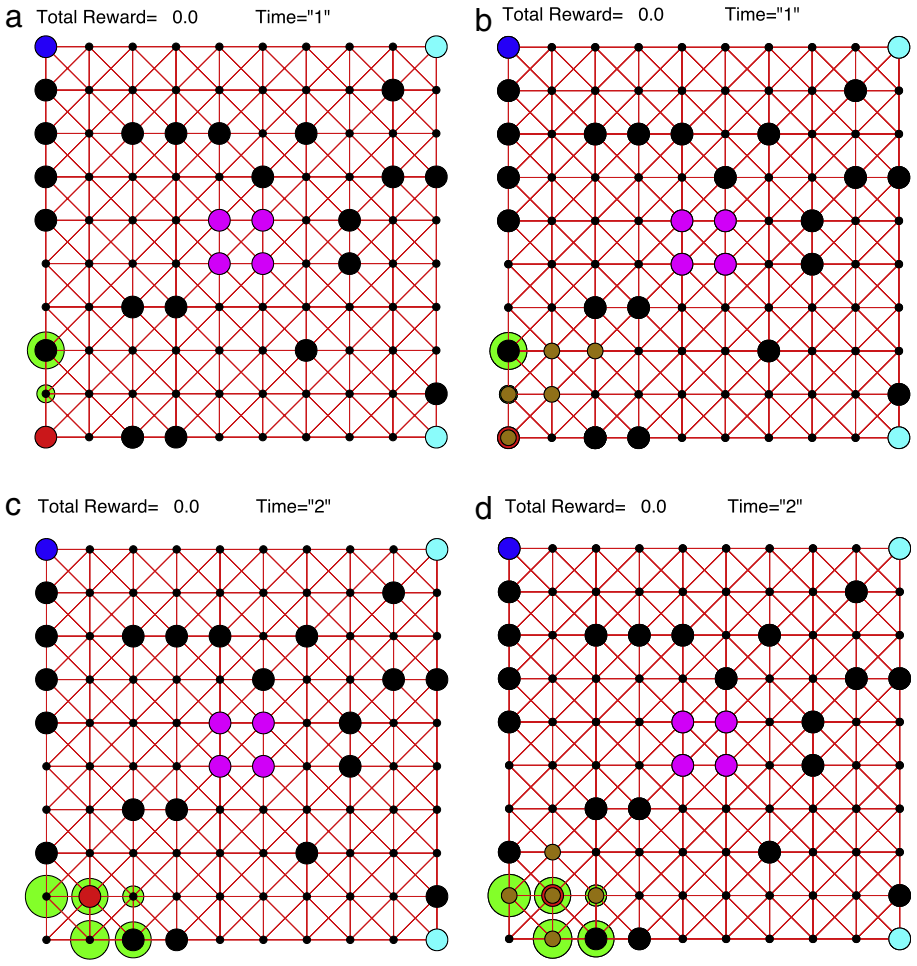


Fig. 4. Snapshots of the system trajectory under the proposed receding horizon control laws. In all snapshots, the states with rewards are marked in green, where the size of the state is proportional with the associated reward. (a) At time $k = 0$, the initial state of the system is marked in red (in the lower left corner). (b) The controller $\mathbf{p}_0^\star = \mathtt{RH}^0(S_{\mathcal{P}0})$ is computed at the initial state. The optimal predicted trajectory $\mathbf{p}_0^\star$ is marked by a sequence of states in brown. (c) The first transition $q_0 \to_{\mathcal{T}} q_1$ is applied on $\mathcal{T}$ and transition $p_0 \to_{\mathcal{P}} p_1$ is applied on $\mathcal{P}$. The current state (q_1) of the system is marked in red. (d) The controller $\mathbf{p}_1^\star = \mathtt{RH}(p_1, \mathbf{p}_0^\star)$ is computed at p_1. The optimal predicted trajectory $\mathbf{p}_1^\star$ is marked by a sequence of states in brown. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

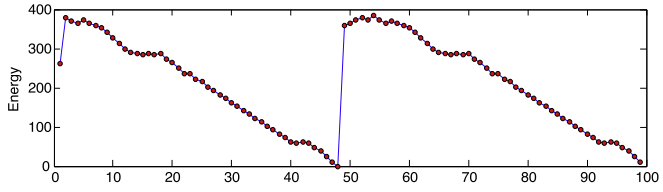


Fig. 5. Plot of energy $V(p)$ at the current state for 100 time-steps (rewards function $\mathcal{R}(q, k)$ can vary at each time-step).

This work can be extended in several directions. The optimization problem of maximizing rewards can be easily extended to other meaningful cost functions. For example, it is possible to assign penalties or costs on states of the system and minimize the accumulated cost of trajectories in the horizon. It is also possible to define costs on state transitions and minimize the control effort. Combinations of the above are also easy to formulate and solve. Our current efforts focus on extending the proposed framework to finite probabilistic systems, such as Markov decision processes and partially observed Markov decision processes, and specifications given as formulas of probabilistic temporal logic. As discussed in Section 4.4, we also aim to extend the results to other finite system models, such as non-deterministic systems.

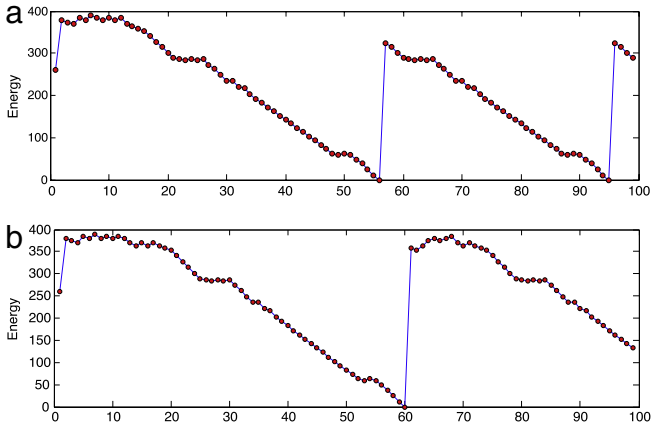


Fig. 6. Plot of energy $V(p)$ at the current state for 100 time-steps: (a) rewards function $\mathcal{R}(q, k)$ varies at every 3 time-steps; (b) rewards function $\mathcal{R}(q, k)$ varies every 5 time-steps.

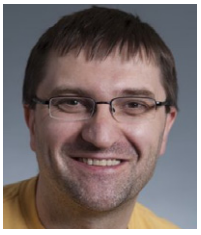
References

Babiak, Tomáš, Křetínský, Mojmír, Řehák, Vojtěch, & Strejček, Jan (2012). LTL to Büchi automata translation: fast and more deterministic. In *Tools and algorithms for the construction and analysis of systems* (pp. 95–109). Springer.

Baier, C., Katoen, J.-P., & Larsen, K. G. (2008). *Principles of model checking*. MIT Press.
Belta, C., Isler, V., & Pappas, G. J. (2005). Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics*, *21*(5), 864–874.
Bemporad, A., & Morari, M. (1999). Control of systems integrating logic, dynamics, and constraints. *Automatica*, *35*, 407–427.
Browne, M. C., Clarke, E. M., & Grumberg, O. (1988). Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, *59*(1–2), 115–131.
Clarke, E. M., Peled, D., & Grumberg, O. (1999). *Model checking*. MIT Press.
Desai, J., Ostrowski, J. P., & Kumar, V. (1998). Controlling formations of multiple mobile robots. In *IEEE international conference on robotics and automation* (pp. 2864–2869).
Di Cairano, S., Lazar, M., Bemporad, A., & Heemels, W. P. M. H. (2008). A control Lyapunov approach to predictive control of hybrid systems. In *Lecture Notes in Computer Science*: *vol. 4981*. *Hybrid systems: computation and control* (pp. 130–143). St. Louis, MO: Springer Verlag.
Ding, Xu Chu, Belta, Calin, & Cassandras, Christos G. (2010). Receding horizon surveillance with temporal logic specifications. In *IEEE conference on decision and control* (pp. 256–261). December.
Ding, Xu Chu, Lazar, Mircea, & Belta, Calin (2012). Receding horizon temporal logic control for finite deterministic systems. In *American control conference*, Montreal, Canada. July.
Gastin, P., & Oddoux, D. (2001). Fast LTL to Buchi automata translation. *Lecture Notes in Computer Science*, 53–65.
Habets, L. C. G. J. M., Collins, P. J., & van Schuppen, J. H. (2006). Reachability and control synthesis for piecewise-affine hybrid systems on simplices. *IEEE Transactions on Automatic Control*, *51*, 938–948.
Habets, L., Kloetzer, M., & Belta, C. (2006). Control of rectangular multi-affine hybrid systems. In *IEEE conference on decision and control* (pp. 2619–2624).
Henzinger, Thomas A., & Piterman, Nir (2006). Solving games without determinization. In *Computer science logic* (pp. 395–410). Springer.
Karaman, S., & Frazzoli, E. (2009). Sampling-based motion planning with deterministic μ-calculus specifications. In *IEEE conference on decision and control* (pp. 2222–2229). Shanghai, China.
Kloetzer, M., & Belta, C. (2008a). A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, *53*(1), 287–297.
Kloetzer, M., & Belta, C. (2008b). Dealing with non-determinism in symbolic control. In M. Egerstedt, & B. Mishra (Eds.), *Lecture notes in computer science*, *Hybrid systems: computation and control* (pp. 287–300). Springer.
Kress-Gazit, H., Fainekos, G., & Pappas, G. J. (2007). Where's Waldo? Sensor-based temporal logic motion planning. In *IEEE international conference on robotics and automation* (pp. 3116–3121).
Li, W., & Cassandras, C. G. (2006). A cooperative receding horizon controller for multivehicle uncertain environments. *IEEE Transactions on Automatic Control*, *51*(2), 242–257.
Lindemann, S. R., Hussein, I. I., & LaValle, S. M. (2007). Real time feedback control for nonholonomic mobile robots with obstacles. In *IEEE conference on decision and control* (pp. 2406–2411). New Orleans, LA. December.
Loizou, S. G., & Kyriakopoulos, K. J. (2004). Automatic synthesis of multiagent motion tasks based on LTL specifications. In *IEEE conference on decision and control* (pp. 153–158). Paradise Islands, The Bahamas. December.
Milner, R. (1989). *Communication and concurrency*. Prentice-Hall.
Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Dover Pubns.
Piterman, N., Pnueli, A., & Saar, Y. (2006). Synthesis of reactive(1) designs. In *International conference on verification, model checking, and abstract interpretation* (pp. 364–380). Charleston, SC. January.
Rawlings, J. B., & Mayne, D. Q. (2009). *Model predictive control: theory and design*. Nob Hill Publishing.
Smith, S. L., Tůmová, J., Belta, C., & Rus, D. (2011). Optimal path planning for surveillance with temporal logic constraints. *International Journal of Robotics Research*, *30*(14), 1695–1708.
Wongpiromsarn, T., Topcu, U., & Murray, R. M. (2009). Receding horizon temporal logic planning for dynamical systems. In *IEEE conference on decision and control and Chinese control conference* (pp. 5997–6004). Shanghai, China. December.

Xuchu Ding received his B.S., M.S. and Ph.D. degree in Electrical and Computer Engineering from the Georgia Institute of Technology, Atlanta, in 2004, 2007 and 2009, respectively. During 2010–2011 he was a Postdoctoral Research Fellow at Boston University. In 2011 he joined United Technologies Research Center as a Senior Research Scientist. His research interests include hierarchical mission planning with formal guarantees under dynamic and rich environments, optimal control of hybrid systems, and coordination of multi-agent networked systems.

Mircea Lazar (born in Iasi, Romania, 1978) received his M.Sc. and Ph.D. in Control Engineering from the Technical University "Gh. Asachi" of Iasi, Romania (2002) and the Eindhoven University of Technology, The Netherlands (2006), respectively. For the Ph.D. Thesis he received the EECI (European Embedded Control Institute) Ph.D. award. Since 2006 he has been an Assistant Professor in the Control Systems group of the Electrical Engineering Faculty at the Eindhoven University of Technology. He is currently serving as Chair of the IEEE CSS Technical Committee on Hybrid Systems and as a Member of the IFAC Technical Committee 2.3 Non-linear control systems. His research interests lie in stability theory, scalable Lyapunov methods, and model predictive control.



Calin Belta is an Associate Professor in the Department of Mechanical Engineering, Department of Electrical and Computer Engineering, and the Division of Systems Engineering at Boston University, where he is also affiliated with the Center for Information and Systems Engineering (CISE), the Bioinformatics Program, and the Center for Biodynamics (CBD). His research focuses on dynamics and control theory, with particular emphasis on hybrid and cyber-physical systems, formal synthesis and verification, and applications in robotics and systems biology. Calin Belta is a Senior Member of the IEEE and an Associate Editor for the SIAM Journal on Control and Optimization (SICON). He received the Air Force Office of Scientific Research Young Investigator Award and the National Science Foundation CAREER Award.