Receding Horizon Surveillance with Temporal Logic Specifications

Xu Chu Ding, Calin Belta, and Christos G. Cassandras

Abstract— **In this paper we consider a setting where a robotic vehicle is commissioned to provide surveillance in an area where there are multiple targets, while satisfying a set of high level, rich specifications expressed as Linear Temporal Logic formulas. Each target has an associated reward. The goal of the vehicle is to maximize the cumulative collected reward while satisfying the given high level task specification. By the nature of a surveillance mission, targets points of interest are detected in real time around the current location of the vehicle; hence we employ a receding horizon controller to compute the optimal path of the vehicle inside a subset of the mission space. This paper provides a framework which guarantees that the overall trajectory of the system satisfies the desired linear temporal logic specification, while the control decisions are made based on local information obtained in real time.**

I. Introduction

Motion planing and control of mobile robots to achieve complex tasks is a fundamental problem in robotics that has been addressed from many perspectives. Traditional motion planning approaches focus on dealing with the complexity of the environment and of the robot dynamics, while restricting the motion specification language to simple primitives of the type "Go from A to B and avoid obstacles" [13]. This is not rich enough to describe a large class of tasks of interest in practical applications. The accomplishment of the mission might require the attainment of either A or B, convergence to a region ("reach A eventually and stay there for all future times"), visiting targets sequentially ("reach A, and then B, and then C"), surveillance ("reach A and then B infinitely often"), or the satisfaction of more complicated temporal and logic conditions about the reachability of regions of interest (*e.g.*, "Never go to A. Don't go to B unless C is visited.").

Recent advances in formal verification and control theory show that it is possible to develop computational frameworks allowing for motion planning and control from rich specifications. First, several abstraction techniques for linear and nonlinear systems ([1], [8], [9], [18], [19]) suggest that the motion of a vehicle in an environment can be represented as a system with finitely many states, such as a finite transition system. Second, it has been shown that the expressivity of temporal logics, such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) [2], is enough to capture a large class of robotic applications [3], [10], [12], [14], [18]. Third, it has been shown that it is possible to generate control strategies for finite transition systems from temporal logic specifications by adapting existing model checking and game-theoretic techniques [11]. Fully automated frameworks for robot deployment from LTL specifications have been developed by combining the three ingredients above [12].

While the works enumerated above bridge some of the gap between control theory, robotics, and formal verification, several questions still remain to be answered. In particular, the connection between optimality, correctness, and locality of information is not well understood. Consider, for example, a scenario in which the motion of a robot in an environment is modeled as a weighted finite transition system, where the weights capture the costs to be paid to travel from one region to another. Assume that rewards of different values can appear and disappear in the regions. Also assume that the robot can only sense rewards in a neighborhood of its current position. From a control point of view, it is desirable to design control strategies that minimize the travel costs and maximize the collected reward. From a formal verification perspective, we want to make sure that the produced trajectories satisfy a temporal logic specification, such as safety or surveillance. The question is how to compromise between these two desiderata, and produce optimal and provably safe control strategies.

In this paper, we provide a partial answer to the above question. Motivated by the abstraction results enumerated above, we assume that the motion of the robot is modeled as a finite and deterministic weighted transition system. We assume that rewards with unknown dynamics appear and disappear at the states and that the robot can only sense these rewards locally. We focus on motion specifications given as LTL formulas over a set of time-invariant properties assigned to the states. We propose a control strategy in the form of an infinite iteration of a receding horizon controller, which is computed based on local reward information. To ensure the satisfaction of the temporal logic specification, we draw inspiration from LTL model checking and construct a product automaton between the transition system and the Büchi automaton generating the language satisfying the LTL formula [2]. Following a procedure that resembles the construction of discrete Lyapunov functions, we propose an algorithm for assignment of costs to the states of the product automaton. These costs are used by the receding horizon controllers in such a way that progress is continuously made towards satisfying the specification (*i.e.,* towards satisfying the Büchi acceptance condition) and at the same time the collection of local rewards is maximized. Our solution is complete in the sense that a control strategy satisfying the LTL specification will always be found if one exists.

This work is an extension of our previous result from

X.C. Ding and C. Belta are with the Department of Mechanical Engineering, C.G. Cassandras is with the Department of Electrical and Computer Engineering, all three authors are also with the Center for Information and Systems Engineering, at Boston University, Brookline, MA 02446, USA. Email: {xcding,cbelta,cgc}@bu.edu. This work is partially supported by the ONR-MURI Award N00014-09-1051 at Boston University.

[12], which can be seen as a particular case of the problem considered here when there is no reward, the robot can sense all the environment states for all times, and the optimization is over the weights of the transitions. It also relates to [3], where a global controller was obtained as a solution of a temporal logic game, for which a winning strategy was guaranteed under some assumptions about environmental events. A receding horizon approach to generate a control from a temporal logic formula was also considered in [18], where a control strategy was computed by dividing the control synthesis problem into smaller sub-problems in a receding horizon like manner. Note that in both these works the specification was restricted to the GR(1) fragment of LTL, whereas here we allow for full LTL expressivity.

Due to space limitations, results in this paper are stated without proof. Proofs of all results can be found in the technical report [5]. Furthermore, [5] analyzes computational complexities of our proposed approach in detail.

II. Preliminaries

Definition 1: A weighted finite transition system (TS) is a tuple $\mathcal{T} = (Q, q_0, \delta, \omega, \Pi, h)$, where Q is the finite set of states; $q_0 \in Q$ is the initial state; $\delta : Q \mapsto 2^Q$ is the transition function; ω is a weight function that assigns positive values to all the transitions (*i.e.,* to all pairs (q_1, q_2) for which $q_2 \in \delta(q_1)$); Π is a finite set of atomic propositions; and $h : Q \mapsto 2^\Pi$ is an output map on the states of $\mathcal{T}$.

Atomic propositions are properties that can be either true or false at each state of the TS. At state q, $\pi \in h(q)$ if and only if π is true at state q. We call q_2 a successor state of q_1 if $q_2 \in \delta(q_1)$. A *trajectory* or *run* of the TS is an infinite sequence $r = q_0 q_1 ...$ where $q_{i+1} \in \delta(q_i)$ for all $i \geq 0$. A run r generates a word $o = o_0 o_1 ...$ where $o_i = h(q_i)$.

We employ Linear Temporal Logic (LTL) to describe high level motion specifications. A detailed description of the syntax and semantics of LTL is beyond the scope of this paper and can be found in [2]. Roughly, an LTL formula is build up from a set of atomic propositions Π, standard Boolean operators $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), and temporal operators $\bigcirc$ (next), $\mathcal{U}$ (until), $\diamond$ (eventually), $\Box$ (always). The semantics of LTL formulas are given over infinite words o in 2^Π, such as the words generated by a TS $\mathcal{T}$. A word satisfies an LTL formula ϕ if ϕ is true at the first position of the word; $\Box\phi$ means that ϕ is true at all positions of the word; $\diamond\phi$ means that ϕ eventually becomes true in the word; $\phi_1 \, \mathcal{U} \phi_2$ 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 are given later). We say a run r satisfies ϕ if and only if the word generated by r satisfies ϕ.

Definition 2: (Büchi Automaton) A Büchi automaton is a tuple $\mathcal{B} = (S_\mathcal{B}, S_{\mathcal{B}0}, \Sigma, \delta_\mathcal{B}, F_\mathcal{B})$, 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; $\delta_\mathcal{B} : S_\mathcal{B} \times \Sigma \to 2^{S_\mathcal{B}}$ is a transition function; and $F_\mathcal{B} \subseteq S_\mathcal{B}$ is the set of accepting states.

A Büchi automaton accepts a word over Σ if at least one of the corresponding runs intersects with $F_\mathcal{B}$ infinitely many times. For any LTL formula ϕ over Π, one can construct a Büchi automaton with input alphabet $\Sigma \subseteq 2^\Pi$ accepting all and only words over Π that satisfy ϕ ([20]). We refer readers to [7], [17] and references therein for efficient algorithms and freely downloadable implementations to translate a LTL formula over Π to a corresponding Büchi automaton $\mathcal{B}$.

Definition 3: (Product Automaton) A weighted product automaton $\mathcal{A} = \mathcal{T} \times \mathcal{B}$ is a tuple $(S_\mathcal{A}, S_{\mathcal{A}0}, \delta_\mathcal{A}, \omega_\mathcal{A}, F_\mathcal{A})$, where $S_\mathcal{A} = Q \times S_\mathcal{B}$ is the set of states; $S_{\mathcal{A}0} = \{q_0\} \times S_{\mathcal{B}0}$ is the set of initial states; $\delta_\mathcal{A} : S_\mathcal{A} \mapsto 2^{S_\mathcal{A}}$ is the transition function defined as $(q_j, s_l) \in \delta_\mathcal{A}((q_i, s_k))$ if and only if $q_j \in \delta(q_i)$ and $s_l \in \delta_\mathcal{B}(s_k, h(q_i))$; $\omega_\mathcal{A} : S_\mathcal{A} \times S_\mathcal{A} \mapsto R^+$ is a positive-valued weight function inherited from $\mathcal{T}$, defined as $\omega_\mathcal{A}((q_i, s_k), (q_j, s_l)) = \omega(q_i, q_j)$, where $(q_j, s_l) \in \delta_\mathcal{A}((q_i, s_k))$; and $F_\mathcal{A} = Q \times F_\mathcal{B}$ is the set of accepting states.

A run $r_\mathcal{A}$ of $\mathcal{A}$ is accepted if and only if $r_\mathcal{A}$ intersects $F_\mathcal{A}$ infinitely many times. We define the projection of a run $r_\mathcal{A}$ onto the TS $\mathcal{T}$ as $\gamma_\mathcal{T}(r_\mathcal{A}) = r = q_0 q_1 ...$, if $r_\mathcal{A} = (q_0, s_0)(q_1, s_1)....$ If ϕ is a LTL formula and $\mathcal{B}$ is its corresponding Büchi automaton, then the projection of an accepted run $r_\mathcal{A}$ on $\mathcal{T}$ is a run of $\mathcal{T}$ that generates a word satisfying ϕ ([6]).

III. Problem Formulation and Approach

In this paper, we assume that the motion of a robot in an environment is modeled as a transition system $\mathcal{T}$ (Def. 1). The set of states Q can be seen as a set of labels for the regions in a partition of the environment. Each transition corresponds to a controller driving the robot between two adjacent regions. The weight function ω can model the maximum time needed to travel between two adjacent regions, the corresponding control effort, or the distance between representative points in the regions.

Such a finite representation of robot motion can be obtained by using popular partition schemes, such as triangulations or rectangular grids, and hierarchical abstractions of dynamical systems. For example, at the first level in the hierarchy, the dynamics of the robot can be mapped from its configuration space (state-space) to its work space, or output space (*e.g.,* position of a representative point of the robot in the environment [4]). At the second level of the hierarchy, feedback controllers for facet reachability and invariance in polytopes ([1], [9]) can be used to construct a finite representation of robot motion in the from of a TS. One can guarantee that the initial dynamical system and the abstraction $\mathcal{T}$ are "equivalent" by ensuring that they are related by simulation and bisimulation relations [15].

Note that our TS $\mathcal{T}$ does not have inputs. In other words, we assume that one can simply choose available transitions at a state. This corresponds to a deterministic TS with inputs, *i.e.,* a system in which the choice of an available input at a state uniquely determines the transition to the next state. The particular control strategy that we develop in this paper is based on this property.

We assume that the motion of the robot in the environment is required to satisfy a rich specification given as an LTL formula over a set of properties (see Sec. II). Note that a

variety of robot surveillance tasks can be easily translated to LTL formulas, *e.g.*: 1) Sequence: "first visit states satisfying a, and then states satisfying b" ($\diamond(a \wedge \diamond b)$) ; 2) Coverage: "visit states satisfying a and states satisfying b, regardless of order" ($\diamond a \wedge \diamond b$); 3) Patrolling: "achieve a sequence task ϕ infinitely many times" ($\square\diamond\phi$); 4) Safety: "achieve task ϕ and always avoid states satisfying c" ($\square\neg c \wedge \phi$).

We assume that rewards (positive real numbers) are associated to some of the states in Q in a time varying fashion. A state with a reward is called a *target*. We do not make any assumptions about the dynamics governing the appearance /disappearance of rewards and the reward values, However, we make the natural assumption that, at each time instant, the robot can only "see" (sense) the rewards in a neighborhood of its current state. We denote by $W(q) \subseteq Q$ the set of targets that the robot can sense when at state $q \in Q$, and by $R(q, q_i), q_i \in W(q)$ the positive awards associated with the targets. Upon visiting a target, the robot collects the corresponding reward, and the state is no longer a target.

Our goal is to generate a robot control strategy such that the produced trajectory maximizes the collected reward while at the same time satisfying the LTL specification. It is important to note that, since we are interested in infinite robot trajectories, it does not make sense to look for a strategy that maximizes the total collected reward, since this can be infinite. Rather, we aim to design a (local and real-time) receding-horizon type controller that maximizes rewards collected based on local information obtained at current state q. Specifically, we consider the following problem:

Problem 1: Given a TS $\mathcal{T}$ and a LTL formula ϕ, design a state-feedback control strategy that (1) maximizes the collected reward from the local target set $W(q)$ and (2) ensures that the produced infinite trajectory satisfies ϕ.

As it will become clear in the next section, the control strategy consists of infinitely many iterations of a state-feedback receding horizon controller. At each state q, the receding horizon controller provides a finite subsequence (which we call a finite run) of a (infinite) run that maximizes the collected reward from the local target set $W(q)$. The control strategy guarantees that the infinite run obtained by repeatedly executing the real time controller and concatenating the finite subsequences satisfies the LTL specification.

In this paper we design a real time control strategy that uses local reward information, maximizes the collected reward locally, and at the same time guarantees satisfaction of ϕ globally. To achieve this, we need a measure of "progress" towards satisfying the formula. If our controller is designed to always further this progress, then we can show that the LTL formula is satisfied. Our approach in this paper is to assign a suitable cost to each state of the product automaton. This cost assignment will be computed off-line once, and then it will be used on-line with the real time controller. The cost assignment will be designed so that if used in conjunction with our proposed control strategy, an accepting state on the product automaton is reached in finite number of transitions. Since this is repeated infinitely many times, the acceptance condition of the product automaton is enforced.

IV. Control Design

In this section, we present the control strategy providing a solution to Prob. 1. The central component of our control strategy is a state feedback controller that produces a finite run on the TS $\mathcal{T}$. Formally, we define a finite run of $\mathcal{T}$ as a finite sequence of states $r_f = q_1 ... q_n$, where $q_{i+1} \in \delta(q_i)$ for all $i = 1, ..., n-1$.

A. Cost Assignment

We first describe our cost assignment scheme. First, given a TS $\mathcal{T}$ and a Büchi automaton $\mathcal{B}$ translated from a LTL formula ϕ, we construct the product automaton $\mathcal{A} = (S_\mathcal{A}, S_{\mathcal{A}0}, \delta_\mathcal{A}, \omega_\mathcal{A}, F_\mathcal{A})$ as defined in Def. 3. Our cost assignment scheme applies a cost as a measure of progress to each state of $\mathcal{A}$.

The notion of distance between states created by the weight function $\omega_\mathcal{A}$ (which is inherited from $\mathcal{T}$) assigned on transitions of $\mathcal{A}$ is a natural choice to be used as the progress metric. A finite run $r_{f\mathcal{A}}$ on $\mathcal{A}$ is defined as a finite subsequence of a run on $\mathcal{A}$, similar to a finite run r_f on $\mathcal{T}$. Now, let $\mathcal{R}(p_i, p_j)$ denote all possible finite runs from a state $p_i \in S_\mathcal{A}$ to a state $p_j \in S_\mathcal{A}$:

$$\begin{aligned}\mathcal{R}(p_i, p_j) \quad = \quad & \{r_{f\mathcal{A}} = p_1 ... p_n | p_1 = p_i, p_n = p_j; \\ & p_{k+1} \in \delta_\mathcal{A}(p_k) \text{ for } k = 1, ..., n-1\}. \quad (1)\end{aligned}$$

Note that $\mathcal{R}(p_i, p_j)$ may not be a finite set due to possible cycles in $\mathcal{A}$, and n in (1) is arbitrary but finite and $n \geq 2$. We say p_i reaches p_j, or p_j is reachable from p_i, if $\mathcal{R}(p_i, p_j)$ is not empty.

Next, we define a path length function with respect to a finite run $r_{f\mathcal{A}} = p_1 ... p_n$ as $L(r_{f\mathcal{A}}) = \sum_{k=1}^{n-1} \omega_\mathcal{A}(p_k, p_{k+1})$. Let $|r_{f\mathcal{A}}|$ represent the number of states in $r_{f\mathcal{A}}$. We set $L(r_{f\mathcal{A}}) = 0$ if $|r_{f\mathcal{A}}| = 1$.

We define a distance function from a state $p_i \in S_\mathcal{A}$ to $p_j \in S_\mathcal{A}$ as follows:

$$d(p_i, p_j) = \begin{cases} \min_{r_{f\mathcal{A}} \in \mathcal{R}(p_i, p_j)} L(r_{f\mathcal{A}}) & \text{if } \mathcal{R}(p_i, p_j) \neq \varnothing \\ \infty & \text{if } \mathcal{R}(p_i, p_j) = \varnothing \end{cases} \quad (2)$$

Since $\omega_\mathcal{A}$ is a positive-valued function, we have $d(p_i, p_j) > 0$ for all $p_i, p_j \in S_\mathcal{A}$. We note that given p_j, $d(p_i, p_j)$ for all $p_i \in S_\mathcal{A}$ can be efficiently computed by several shortest path algorithms [16], such as Dijkstra's algorithm.

We say that a set $A \subseteq S_\mathcal{A}$ is *self-reachable* if and only if all states in A can reach a state in A ($\forall p_i \in A$, $\exists p_j \in A$ such that $\mathcal{R}(p_i, p_j) \neq \varnothing$). We define $F_{\mathcal{A}^\star}$ to be the largest self-reachable subset of $F_\mathcal{A}$. Now, using (2), we propose the following cost function defined on all states $p_i \in S_\mathcal{A}$:

$$J(p_i) = \begin{cases} \min_{p_j \in F_{\mathcal{A}^\star}} d(p_i, p_j), & \text{if } p_i \notin F_{\mathcal{A}^\star} \\ 0, & \text{if } p_i \in F_{\mathcal{A}^\star} \end{cases} \quad (3)$$

Clearly, $J(p_i) = 0$ if and only if $p_i \in F_{\mathcal{A}^\star}$. This cost encodes the minimum distance from states in $\mathcal{A}$ to the set $F_{\mathcal{A}^\star}$.

We propose Alg. 1 to obtain the set $F_{\mathcal{A}^\star}$ and the cost measure J. This algorithm obtains the largest self-reachable subset of $F_\mathcal{A}$ by construction, because it starts with the whole set $F_\mathcal{A}$ and prunes out one by one states that can not reach a

state in itself, until all states in the set satisfies the definition of a self-reachable set.

Algorithm 1 Cost assignment algorithm, given a product automaton $\mathcal{A} = (S_{\mathcal{A}}, S_{\mathcal{A}0}, \delta_{\mathcal{A}}, \omega_{\mathcal{A}}, F_{\mathcal{A}})$

1: Compute $d(p_i, p_j)$ for all $p_i \in S_{\mathcal{A}}$ and $p_j \in F_{\mathcal{A}}$.
2: Set $F_{\mathcal{A}^\star} = F_{\mathcal{A}}$.
3: **while** there exist $q \in F_{\mathcal{A}^\star}$ such that

$$\min_{p_j \in F_{\mathcal{A}^\star}, p_i \in \delta_{\mathcal{A}}(q)} d(p_i, p_j) = \infty$$

do
4: Remove q from $F_{\mathcal{A}^\star}$.
5: **end while**
6: Obtain $J(p_i)$ using the definition (3) for all $p_i \in S_{\mathcal{A}}$.

Alg. 1 is run only once off-line (before deployment). The costs $J(p_i)$ are saved in a look-up table to be used later in conjunction with the real-time controllers.

Lemma 1: All accepting states in an accepted run $r_{\mathcal{A}}$ of $\mathcal{A}$ must be in $F_{\mathcal{A}^\star}$.

The corollary below follows directly from Lemma 1 and the definition of J in (3) and $F_{\mathcal{A}^\star}$.

Corollary 1: An accepted run $r_{\mathcal{A}}$ of $\mathcal{A}$ originating at p exists if and only if $J(p) \neq \infty$. Furthermore, all states in an accepted run $r_{\mathcal{A}}$ have finite cost.

As a consequence, by looking up the cost assignment J, we find initial states of $\mathcal{T}$ from which there do not exist a satisfying run, where all states in $S_{\mathcal{A}0}$ have infinite cost.

The following lemma is another property of the cost function which will be useful later.

Lemma 2: If $p_i \in \mathcal{A}$ and $J(p_i) \in (0, \infty)$, then there is at least one state $p_j \in \delta_{\mathcal{A}}(p_i)$, such that $J(p_j) < J(p_i)$.

B. Receding Horizon Controllers

In this sub-section we present the controllers $P^{init}_{\mathcal{A}}(S_{\mathcal{A}0})$ and $P_{\mathcal{A}}(p)$ that produce finite runs $r_{f\mathcal{A}}$ on $\mathcal{A}$, assuming that the cost J defined in (3) is accessible from a look-up table. The feedback controller we use to produce solutions to Prob. 1 is obtained as the projection of $P^{init}_{\mathcal{A}}(S_{\mathcal{A}0})$ (if current state is the initial state) or $P_{\mathcal{A}}(p)$ (if otherwise) onto $\mathcal{T}$. The projection function for finite runs, $\gamma_{\mathcal{T}}(r_{f\mathcal{A}}) = r_f$, is defined in the same way as in the case of infinite runs.

We design $P_{\mathcal{A}}(S_{\mathcal{A}0})$ and $P_{\mathcal{A}}(p)$ as receding horizon controllers in the sense that they maximize the reward collected over finite runs with path length no more than a pre-defined "planning horizon length" T. Since the weight function $\omega_{\mathcal{A}}$ is inherited from $\mathcal{T}$, the projection of the resultant optimal finite run onto $\mathcal{T}$ also has path length no more than T. We choose T large enough such that $T \geq \max_{p_i, p_j \in S_{\mathcal{A}}, p_j \in \delta_{\mathcal{A}}(p_i)} \omega_{\mathcal{A}}(p_i, p_j)$, which implies that no single edge-weight is higher than the horizon length.

First we propose the controller $P_{\mathcal{A}}(p)$ for a given state $p = (q, s)$ in $\mathcal{A}$. We aim to design $P_{\mathcal{A}}(p)$ such that repeated executions of the controller produce an accepted run on $\mathcal{A}$ originating from p. The idea to achieve this is the following: we design a "pre controller" $P^{pre}_{\mathcal{A}}(p)$ and let $P_{\mathcal{A}}(p) = P^{pre}_{\mathcal{A}}(p)$ if $J(p) \in (0, \infty)$. Repeated executions of $P^{pre}_{\mathcal{A}}(p)$ drive the system from any state p on $\mathcal{A}$, where $J(p) \in (0, \infty)$, to a state in $F_{\mathcal{A}^\star}$ in a finite number of transitions, while maximizing the reward collected locally. We also design a "post controller" $P^{post}_{\mathcal{A}}(p)$ and let $P_{\mathcal{A}}(p) = P^{post}_{\mathcal{A}}(p)$ if $J(p) = 0$. $P^{post}_{\mathcal{A}}(p)$ drives the system from a state in $F_{\mathcal{A}^\star}$ to a state $p_i \in \mathcal{A}$ such that $J(p_i) \neq \infty$. At this point if $J(p_i) > 0$, then the pre controller takes over and again drives the system to a state in $F_{\mathcal{A}^\star}$ in finite number of transitions. If $J(p_i) = 0$, then the post controller is executed again. This process repeats infinitely many times so that the set $F_{\mathcal{A}^\star}$ (which is a subset of $F_{\mathcal{A}}$) is visited infinitely many times, and thus the acceptance condition is enforced.

To begin, we define a cumulative reward function on a finite run $r_{f\mathcal{A}}$ as $\Re(r_{f\mathcal{A}}, W(q)) = \sum_{q_i \in \tau(r_{f\mathcal{A}}, W(q))} R(q, q_i)$, where $\tau(r_{f\mathcal{A}}, W(q))$ is the set:

$$\tau(r_{f\mathcal{A}}, W(q)) = \{q_i | q_i \in W(q) \text{ and } q_i \text{ is in } \gamma_{\mathcal{T}}(r_{f\mathcal{A}})\}.$$

It is apparent that $\Re(r_{f\mathcal{A}}, W(q)) \geq 0$ and is finite. Then, we define the pre controller to be:

$$\begin{aligned} P^{pre}_{\mathcal{A}}(p) \quad = \quad & \underset{r_{f\mathcal{A}} = p_1 \ldots p_n}{\arg\max} \ \Re(r_{f\mathcal{A}}, W(q)), \\ & \text{subject to: } p_1 \in \delta_{\mathcal{A}}(p), J(p_n) < J(p), \\ & \omega_{\mathcal{A}}(p, p_1) + L(r_{f\mathcal{A}}) \leq T. \end{aligned} \tag{4}$$

Remark 1: Finding a solution for (4) involves exploring and searching finite runs in a sub-graph of the graph corresponding to the product automaton $\mathcal{A}$. The states of this sub-graph are $\{p_i \in S_{\mathcal{A}} | d(p, p_i) \leq T\}$, since we do not consider finite runs with the last state further than T away from the current state p. This fact demonstrates the local nature of our controller, as the control decision $r_{f\mathcal{A}}$ is always made on this sub-graph (depending on T and p). As a result, to compute $P^{pre}_{\mathcal{A}}(p)$, only a local portion of the product automaton $\mathcal{A}$ around p needs to be constructed and explored. The same remark can be made later for $P^{post}_{\mathcal{A}}(p)$ and $P^{init}_{\mathcal{A}}(S_{\mathcal{A}0})$.

The optimal run computed by $P^{pre}_{\mathcal{A}}(p)$ only requires that the cost on the last state of the finite run to be lower than the current state. This allows the controller to produce runs that go "out of the way" to grab rewards associated with targets.

Before we can use $P^{pre}_{\mathcal{A}}(p)$ in the control algorithm, we have to first verify that it always has a solution.

Lemma 3: Optimization problem $P^{pre}_{\mathcal{A}}(p)$ always has at least one solution for all p where $J(p) \in (0, \infty)$.

Next we show that the pre controller drives a state on $\mathcal{A}$ to one in $F_{\mathcal{A}^\star}$ in a finite number of transitions. We first define a concatenation operator $\oplus$ for finite runs $r^1_{f\mathcal{A}} = p^1_1 \ldots p^1_{n_1}$ and $r^2_{f\mathcal{A}} = p^2_1 \ldots p^2_{n_2}$ such that, if $p^2_1 \in \delta_{\mathcal{A}}(p^1_{n_1})$, then $r^1_{f\mathcal{A}} \oplus r^2_{f\mathcal{A}}$ is a finite run $p^1_1 \ldots p^1_{n_1} p^2_1 \ldots p^2_{n_2}$, and undefined if $p^2_1 \notin \delta_{\mathcal{A}}(p^1_{n_1})$. We also use the notation $\oplus^N_{i=1} r^i_{f\mathcal{A}}$ to indicate repeated concatenation of N finite runs (N could be ∞). Then, we define K (K finite) repeated executions of $P^{pre}_{\mathcal{A}}(p)$ as $P^{pre}_{\mathcal{A}K}(p) = \oplus^K_{i=1} P^{pre}_{\mathcal{A}}(p^{i-1}_{n_{i-1}})$, where $p^0_{n_0} = p$ and $P^{pre}_{\mathcal{A}}(p^{i-1}_{n_{i-1}}) = p^i_1 \ldots p^i_{n_i}$, assuming that $J(p^{i-1}_{n_{i-1}}) > 0$ for $i = 1, \ldots, K$. We then provide the following theorem:

Theorem 1: Given p such that $J(p) \in (0, \infty)$, there exists K bounded above by a constant such that $P^{pre}_{\mathcal{A}K}(p)$ is a finite run ending at a state $p^K_{n_K}$ where $J(p^K_{n_K}) = 0$.

Now we propose a receding horizon controller $P^{post}_{\mathcal{A}}(p)$ that computes a control decision when $p \in F_{\mathcal{A}^\star}$ (thus $J(p) = 0$). To ensure satisfaction of the acceptance condition of $\mathcal{A}$, we must drive the state to one where the cost is finite (such a state must exist by definition of $F_{\mathcal{A}^\star}$). Hence, we have:

$$\begin{aligned} P^{post}_{\mathcal{A}}(p) \quad = \quad & \underset{r_{f\mathcal{A}}=p_1 \ldots p_n}{\arg\max} \ \Re(r_{f\mathcal{A}}, W(q)), \\ & \text{subject to: } p_1 \in \delta_{\mathcal{A}}(p), J(p_n) \neq \infty, \\ & \omega_{\mathcal{A}}(p, p_1) + L(r_{f\mathcal{A}}) \leq T. \end{aligned} \tag{5}$$

Since $J(p) = 0$, $P^{post}_{\mathcal{A}}(p)$ always has a solution because there exist $q_i \in \delta_{\mathcal{A}}(p)$ where $J(q_i) \neq \infty$.

Finally, we propose the controller $P^{init}_{\mathcal{A}}(S_{\mathcal{A}0})$ which is executed for the initial state q_0. This controller takes the input of the set $S_{\mathcal{A}0}$ because the Büchi automaton $\mathcal{B}$ can have a set of initial states $S_{\mathcal{B}0}$. As a result, we can pick the initial state of $\mathcal{A}$ from the set $S_{\mathcal{A}0} = \{q_0\} \times S_{\mathcal{B}0}$. Define $S^\star_{\mathcal{A}0}$ to be $S^\star_{\mathcal{A}0} = \{p_i \in S_{\mathcal{A}0} | J(p_i) \neq \infty\}$. We define:

$$\begin{aligned} P^{init}_{\mathcal{A}}(S_{\mathcal{A}0}) \quad = \quad & \underset{r_{f\mathcal{A}}=p_1 \ldots p_n}{\arg\max} \ \Re(r_{f\mathcal{A}}, W(q_0)), \\ & \text{subject to: } \exists p_0 \in S^\star_{\mathcal{A}0} \text{ s.t. } p_1 \in \delta_{\mathcal{A}}(p_0), \\ & J(p_n) \neq \infty, \omega_{\mathcal{A}}(p_0, p_1) + L(r_{f\mathcal{A}}) \leq T. \end{aligned} \tag{6}$$

If $S^\star_{\mathcal{A}0}$ is empty, due to Corollary 1, there does not exist a run on $\mathcal{T}$ starting at q_0 satisfying ϕ. If $S^\star_{\mathcal{A}0}$ is not empty, then $P^{init}_{\mathcal{A}}(S_{\mathcal{A}0})$ always has a solution due to Lemma 2.

C. Control Algorithm

The overall control strategy for the TS $\mathcal{T}$ is given in Alg. 2. After some off-line computation (*i.e.,* before deployment), the algorithm generates an on-line, receding horizon controller $P(q)$, which is a map from Q to the set of finite runs over Q such that $P(q) = q_1 \ldots q_n$, where q denotes the current state of $\mathcal{T}$ and $q_1 \in \delta(q)$, means that the robot needs to visit the sequence of states $q_1 \ldots q_n$.

Remark 2: In the on-line portion of Alg. 2, we always update the target node set $W(q)$ before we make a control decision (see lines 3 and 5). We can increase the update frequency of $W(q)$, or design the algorithm so that the controller recomputes the control decision when $W(q)$ changes. These alternative choices of control strategies may offer better performance in terms of reward collection, but they also have increased computational complexities.

Finally, we show that Alg. 2 always implements a run r satisfying the given specification ϕ, assuming such a run from q_0 exists. Thus, Prob. 1 is solved.

Theorem 2: Assume that there exists a satisfying run originating from q_0 for a TS $\mathcal{T}$ and a LTL formula ϕ. Then, the application of Alg. 2 to $\mathcal{T}$ and ϕ results in a run $r = q_0 q_1, \ldots$ of $\mathcal{T}$ satisfying ϕ.

Algorithm 2 Control algorithm for $\mathcal{T} = (Q, q_0, \delta, \omega, \Pi, h)$, given a LTL formula ϕ

Executed Off-line:

1: Construct a Büchi automaton $\mathcal{B} = (S_{\mathcal{B}}, S_{\mathcal{B}0}, \Sigma, \delta_{\mathcal{B}}, F_{\mathcal{B}})$ corresponding to ϕ.
2: Construct the product automaton $\mathcal{A} = \mathcal{T} \times \mathcal{B} = (S_{\mathcal{A}}, S_{\mathcal{A}0}, \delta_{\mathcal{A}}, \omega_{\mathcal{A}}, F_{\mathcal{A}})$.
3: Execute Alg. 1 to compute $J(p_i)$ for all $p_i \in S_{\mathcal{A}}$.

Executed On-line:

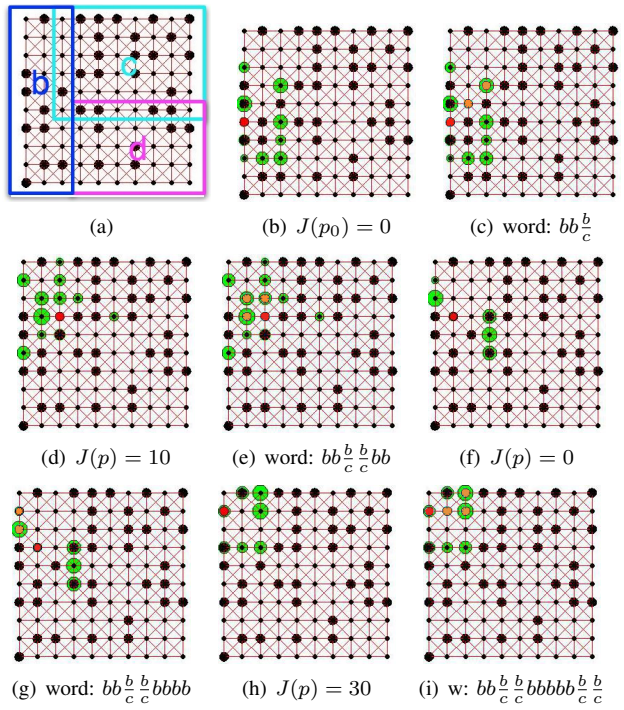
1: **if** $\exists p_0 \in S_{\mathcal{A}0}$ such that $J(p_0) \neq \infty$ **then**
2: Update target set $W(q_0)$ and rewards $R(q_0, q_i)$.
3: Obtain $P^{init}_{\mathcal{A}}(S_{\mathcal{A}0}) = r_{f\mathcal{A}} = p_1 \ldots p_n$. Set $P(q_0) = \gamma_{\mathcal{T}}(P^{init}_{\mathcal{A}}(S_{\mathcal{A}0})) = r_f = q_1 \ldots q_n$. Control $\mathcal{T}$ to implement r_f. Set $q = q_n$, $p = p_n$.
4: **loop**
5: Update target set $W(q)$ and rewards $R(q, q_i)$.
6: Let $P_{\mathcal{A}}(p) = P^{pre}_{\mathcal{A}}(p)$ if $J(p) > 0$ and $P_{\mathcal{A}}(p) = P^{post}_{\mathcal{A}}(p)$ if $J(p) = 0$. Obtain $P_{\mathcal{A}}(p) = r_{f\mathcal{A}} = p_1 \ldots p_n$. Set $P(q) = \gamma_{\mathcal{T}}(P_{\mathcal{A}}(p)) = r_f = q_1 \ldots q_n$. Control $\mathcal{T}$ to implement r_f. Set $q = q_n$, $p = p_n$.
7: **end loop**
8: **else**
9: There is no run originating from q_0 that satisfies ϕ.
10: **end if**

V. Example

In this section we demonstrate our proposed algorithm with an example. For this example, $\mathcal{T}$ has 100 states, which are at the vertices of a rectangular grid with cell size 10 (see Fig. 1(a)). The distance function ω measures the Euclidean distance between states, and there is a transition between two states if the distance between them is less than or equal to 15. We assume that the robot can "sense" targets within a disk of radius 30 centered at its position. We randomly generate the set of targets $W(q)$ within this neighborhood from a uniform distribution for every iteration of Alg. 2. The reward values for the targets in $W(q)$ are generated by uniform sampling from the range 10-25. We choose the horizon length $T = 30$.

We assume that the set of atomic propositions is $\Pi = \{a, b, c, d\}$. The assignment of atomic propositions to the states of $\mathcal{T}$ is shown in Fig. 1(a) and the corresponding caption. Note that some states satisfy more than one atomic proposition. We consider the following safe surveillance specification: "Avoid states satisfying a for all times; visit states satisfying b and then states satisfying either c or d, infinitely often." The specification translates to the following LTL formula: $\phi = \Box\neg a \wedge \Box(\Diamond(b \wedge \Diamond c) \vee \Diamond(b \wedge \Diamond d))$.

Some snapshots from the robot run produced by applying the method developed in this paper are shown in Figs. (1(b) - 1(i)). From these figures, we can see that the trajectory of the robot is satisfying the surveillance specification and maximizing rewards collected locally. In Fig. 2, $J(p)$ at the beginning of each loop in Alg. 2 are shown for about 70 iterations of loops. As expected, states in $F_{\mathcal{A}^\star}$ (which are accepting states with 0 cost) are repeatedly reached.



(a) (b) $J(p_0) = 0$ (c) word: $bb\frac{b}{c}$

(d) $J(p) = 10$ (e) word: $bb\frac{b}{c}\frac{b}{c}bb$ (f) $J(p) = 0$

(g) word: $bb\frac{b}{c}\frac{b}{c}bbbb$ (h) $J(p) = 30$ (i) w: $bb\frac{b}{c}\frac{b}{c}bbbbb\frac{b}{c}\frac{b}{c}$

Fig. 1. Robot deployment corresponding to the given specification ϕ. The TS $\mathcal{T}$ and the assignment of atomic propositions to its states are shown in (a): for b, c, and d, the states inside a rectangle are assigned the atomic propositions with the corresponding color; the states satisfying a are plotted with a bigger radius than other states. Figures (b)-(i) show snapshots from the deployment, where (b), (d), (f), and (h) show the positions of the robot (red) corresponding to updates of $W(q)$ and $R(q, q_i)$ and (c), (e), (g), and (i) show the corresponding optimal finite runs (orange). In all the snapshots, the targets are shown in green circles, where the radii of the circles are proportional to the associated rewards.

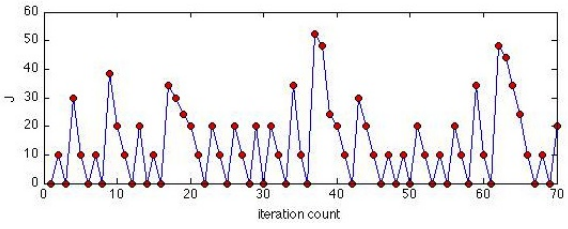


Fig. 2. Cost of the current state of $\mathcal{A}$ ($J(p)$) at the beginning of each iteration of Alg. 2 for 70 iterations.

In terms of computation time, the translation of the formula to a Büchi automaton, the construction of the product automaton, and the cost assignment on the product automaton took 2 sec, 1 sec, and 21 sec, respectively. Each loop in Alg. 2 took about 2 sec. All computation was performed on a MacBook Pro with a dual core processor at 2.5 GHz.

VI. Conclusions and Final Remarks

In this paper, we considered a scenario in which a robot with limited sensing capabilities is required to satisfy temporal logic motion specifications globally, while at the same time maximizing the reward collected from targets that are sensed locally. We developed a control strategy that makes local control decisions in terms of maximizing the reward while ensuring eventual satisfaction of the specification.

For future work, we plan to accommodate more realistic scenarios than the one considered in this paper. Specifically, we will investigate the use of Markov Decision Processes (MDP) and Partially Observable Markov Decision Processes (POMDP) to model the robot motion and of probabilistic temporal logic such as probabilistic LTL (PLTL) and probabilistic CTL (PCTL) as specification languages.

Acknowledgements

We would like to thank Jiri Barnat and Jana Tumova for their useful discussions on LTL model checking.

References

[1] C. Belta and L.C.G.J.M. Habets. Controlling a class of nonlinear systems on rectangles. *IEEE Transactions on Automatic Control*, 51(11):1749 – 1759, 2006.

[2] E. M. M. Clarke, D. Peled, and O. Grumberg. *Model checking*. MIT Press, 1999.

[3] D.C. Conner, H. Kress-Gazit, H. Choset, A.A. Rizzi, and G.J. Pappas. Valet parking without a valet. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, pages 572–577, 2007.

[4] J. Desai, J.P. Ostrowski, and V. Kumar. Controlling formations of multiple mobile robots. In *Proc. IEEE Int. Conf. Robot. Automat.*, Leuven, Belgium, 1998.

[5] Xu Chu Ding, Calin Belta, and Christos G. Cassandras. Receding horizon surveillance with temporal logic specifications. Technical report, Boston University, 2010. (available online at http://hyness.bu.edu/xcdingCDC10/xcdingTechreport2010.pdf).

[6] G.E. Fainekos, H. Kress-Gazit, and G.J. Pappas. Hybrid controllers for path planning: A temporal logic approach. In *IEEE Conference on Decision and Control*, volume 44, page 4885. Citeseer, 2005.

[7] P. Gastin and D. Oddoux. Fast LTL to Buchi automata translation. *Lecture Notes in Computer Science*, pages 53–65, 2001.

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

[9] L.C.G.J.M. Habets, P.J. Collins, and J.H. van Schuppen. Reachability and control synthesis for piecewise-affine hybrid systems on simplices. *IEEE Trans. Aut. Control*, 51:938–948, 2006.

[10] S. Karaman and E. Frazzoli. Vehicle routing problem with metric temporal logic specifications. *47th IEEE Conference on Decision and Control, 2008. CDC 2008*, pages 3953–3958, 2008.

[11] M. Kloetzer and C. Belta. Dealing with nondeterminism in symbolic control. In M. Egerstedt and B. Mishra, editors, *Hybrid Systems: Computation and Control: 11th International Workshop*, Lecture Notes in Computer Science, pages 287–300. Springer Berlin / Heidelberg, 2008.

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

[13] S.M. LaValle. *Planning algorithms*. Cambridge Univ Pr, 2006.

[14] S.G. Loizou and K.J. Kyriakopoulos. Automatic synthesis of multi-agent motion tasks based on LTL specifications. In *43rd IEEE Conference on Decision and Control, 2004. CDC*, 2004.

[15] R. Milner. *Communication and concurrency*. Prentice-Hall, 1989.

[16] C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Dover Pubns, 1998.

[17] F. Somenzi and R. Bloem. Efficient büchi automata from ltl formulae. *Twelfth Conference on Computer Aided Verification (CAV'00)*, pages 248–263, 2000.

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

[19] P. Tabuada. Symbolic control of linear systems based on symbolic subsystems. *IEEE Transactions on Automatic Control*, 51(6):1003–1013, 2006.

[20] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.