Information-guided persistent monitoring under temporal logic constraints

Austin Jones, Mac Schwager, and Calin Belta

***Abstract*—We study the problem of planning the motion of an agent such that it maintains indefinitely a high-quality estimate of some *a priori* unknown feature, such as traffic levels in an urban environment. Persistent operation requires that the agent satisfy motion constraints, such as visiting charging stations infinitely often, which are readily described by rich linear temporal logic (LTL) specifications. We propose and evaluate via simulation a two-level dynamic programming algorithm that is guaranteed to satisfy given LTL constraints. The low-level path planner implements a receding horizon algorithm that maximizes the local information gathering rate. The high-level planner selects inputs to the low-level planner based on global performance considerations.**

I. Motivation

In this paper, we address the problem of planning the path of a mobile robot such that it persistently maintains a high-quality estimate of some unknown feature, i.e. persistent monitoring. This describes many real-world applications in which human decision-makers require real-time information about large environments, such as forest rangers monitoring wildfires or traffic engineers monitoring congestion. In order to provide up-to-date information reliably, the robot has to plan its motion such that relevant information is gained, e.g. the agent visits locations that have not recently been visited. This can be formalized as maximizing the expected mutual information rate (MIR) between the environment and the agent's sensor measurements. In addition, the agent's motion must be planned such that it can continue functioning indefinitely, i.e. it must satisfy constraints such as "Regularly visit a recharging station and always avoid obstacles". In this work, we use linear temporal logic (LTL), an extension of Boolean logic that is capable of describing how a system may change over time, to model such constraints. We present a hierarchal stochastic optimal control algorithm that is guaranteed to satisfy the given LTL constraints while attempting to maximize the MIR. The low-level planner executes a receding horizon algorithm that maximizes the MIR locally. The high-level planner uses a pre-computed policy to select inputs to the receding horizon algorithm such that the LTL constraints are guaranteed to be met and to maximize the MIR over the long horizon. Our procedure is evaluated via Monte Carlo simulation.

Persistent monitoring strategies have recently been considered in the literature. In [21], the authors demonstrate how to regulate the speed of an agent moving along a fixed path such that the uncertainty about the state of the environment remains below a certain threshold indefinitely. This work is extended to planning trajectories in [16]. The problem of multi-agent persistent monitoring is investigated in [6]. The above works consider a measurement model that is independent of the agent's location in the environment and either completely unconstrained motion or motion along predefined paths. In contrast, we assume that the agent's sensing capability depends on its position in the environment and we incorporate linear temporal logic (LTL) motion constraints.

Linear temporal logic [2], [22] can be used to ensure that a system satisfies liveness ("Visit a charging station infinitely often"), fairness ("Ensure a data upload station is visited before visiting a charging station"), and safety ("Always avoid obstacles") properties. Off-the-shelf formal synthesis tools exist for planning a robot's trajectory such that it is guaranteed to satisfy a given LTL formula [1], [5], [23]. These tools were applied to the problem of persistent monitoring in [9] in which the goal was to minimize the amount of time between visiting pre-specified surveillance regions. This work did not explicitly represent the agent's sensing model nor the environment model.

We previously incorporated information-based planning with temporal logic constraints in [10], [11]. In [11], we considered planning under syntactically co-safe linear temporal logic (scLTL) constraints on the motion of the robot, a finite-time subset of LTL. scLTL can be used to describe reachability and finite-time safety properties, among others. By considering full LTL formulae, the approach in this paper allows us to consider infinite-horizon properties such as visiting different sequences

Austin Jones is with the Division of Systems Engineering, Mac Schwager and Calin Belta are with the Division of Systems Engineering and the Department of Mechanical Engineering at Boston University, Boston, MA 02115. Email: {austinmj,schwager,cbelta}@bu.edu

This work was partially supported by ONR under grants MURI N00014-09-1051 and ONR MURI N00014-10-10952, and N00014-12-1-1000, and by NSF under grant CNS-1035588

of regions in the environment infinitely often.

The algorithm in this paper extends the receding-horizon planner developed in [11] to guarantee satisfaction of LTL constraints. In addition, to avoid myopia inherent in receding horizon implementations, we develop a high-level sample-based planner that selects inputs to the receding horizon algorithm that attempt to maximize the MIR over the infinite horizon. An implementation of our procedure is applied to a simulation of a target tracking case study.

II. Mathematical Preliminaries

For sets A and B, 2^A denotes the power set of A, $A \times B$ is the Cartesian product of A and B, and $A^n = A \times A \times \dots A$. We use the shorthand notation $x^{1:t}$ for a time-indexed sequence $x^1 \dots x^t$. The set of all finite and set of all infinite words over alphabet Σ are denoted by Σ^* and Σ^∞, respectively.

We denote the *conditional entropy* between random variables X and Y as $H(X|Y) = H(p_X|p_Y) = -\sum_y \sum_x p_{X,Y}(x,y) \log(p_{X|Y}(x|y))$. We denote the *mutual information* between random variables X and Y as $I(X;Y) = H(X|Y) - H(X)$. Roughly speaking, the mutual information $I(X;Y)$ is the increase in certainty about the state of X when Y is known [19].

A *deterministic transition system* [2] is a tuple $TS = (Q, q_0, Act, Trans, AP, L)$, where Q is a set of states, $q_0 \in Q$ is the initial state, Act is a set of actions, $Trans \subseteq Q \times Act \times Q$ is a transition relation, AP is a set of atomic propositions, and $L : Q \to 2^{AP}$ is a labeling function of states to atomic propositions.

A *discrete time Markov Chain* (MC) is a tuple $MC = (S, s_0, P)$ where S is a discrete set of states, $P : S \times S \to [0,1]$ is a probabilistic transition relation such that the chain moves from state s to state s' with probability $P(s,s')$, and s_0 is an initial state of the system.

A *discrete time Markov decision process* (MDP) is a tuple $MDP = (S, s^0, P, Act)$, where S, s^0 are as defined for a MC, Act is a set of actions, and $P : S \times Act \times S \to [0,1]$ is a probabilistic transition relation such that taking action a drives MDP from state s to state s' with probability $P(s,a,s')$. We denote the set of actions a that can be taken at state s such that $\exists s' \in S$ with $P(s,a,s') > 0$ as $Act(s) \subseteq Act$. A *sample path* of an MDP is a sequence of states $s^0 s^1 \dots \in S^\infty$ with $P(s^i, a, s^{i+1}) > 0$ for some $a \in Act(s^i)$ $\forall i \in \mathbb{N}$.

In this paper, we assume that the reader is familiar with dynamic programming and stochastic optimal control over MDPs [4].

A *linear temporal logic* (LTL) formula is inductively defined as follows [14]:

$$\phi := p | \neg\phi | \phi \vee \phi | \phi \wedge \phi | \phi \ \mathcal{U} \ \phi | \ \bigcirc \ \phi | \ \Diamond \ \phi | \ \Box \ \phi, \quad (1)$$

where p is an atomic proposition, $\neg$ (negation), $\vee$ (disjunction), and $\wedge$ (conjunction) are Boolean operators, and $\bigcirc$ ("next"), $\mathcal{U}$ ("until"), $\Diamond$ ("eventually"), and $\Box$ ("always") are temporal operators. LTL allows for the specification of persistent performance requirements, such as "Obstacles are always avoided" or "Data is uploaded infinitely often".

A *deterministic Buchi automaton* (DBA) [2] is a tuple $\mathcal{A} = (\Sigma, \Pi, \delta, \sigma_0, F)$ where Σ is a finite set of states, Π is a finite alphabet, $\delta \subseteq \Sigma \times \Pi \times \Sigma$ is a deterministic transition relation, $\sigma_0 \in \Sigma$ is the initial state of the automaton, and $F \subseteq \Sigma$ is a set of final (accepting) states. An *accepting run* on a DBA is a sequence of states in Σ^∞ that intersects with F infinitely often. The *language* of a DBA (written $\mathcal{L}(\mathcal{A})$) is the set of all accepting words in Σ^∞. Given an LTL formula ϕ, there exist algorithmic procedures to construct a DBA $\mathcal{A}_\phi$ with alphabet 2^{AP} such that the language of all words satisfying ϕ, $\mathcal{L}(\phi)$, is equal to $\mathcal{L}(\mathcal{A}_\phi)$. [22].

The *product automaton* of a transition system $TS = (Q, q_0, Act, Trans, AP, L)$ and a DBA $\mathcal{A}_\phi = (\Sigma, 2^{AP}, \delta, \sigma_0, F)$ is the Buchi automaton $\mathcal{P} = TS \times \mathcal{A}_\phi = (\Sigma_\mathcal{P}, \chi^0, Act, F_\mathcal{P}, \Delta_\mathcal{P})$ [2]. $\Sigma_\mathcal{P} \subseteq Q \times \Sigma$ is the state space of the automaton, $\chi^0 = (q_0, \sigma_0)$ is the initial state, and $F_\mathcal{P} \subseteq Q \times F$ is the set of accepting states. The transition relation is defined as $\Delta_\mathcal{P} = \{(q,\sigma), p, (q',\sigma') | (q,p,q') \in Trans, (\sigma, L(q), \sigma') \in \delta\}$. The state of the automaton at time k, (q^k, σ^k) is denoted as χ^k for short. If $\chi^{0:k}$ satisfies the acceptance condition on $\mathcal{P}$, then the trajectory $q^{0:k}$ satisfies ϕ.

We define the *distance to acceptance* [1] as a function $W : \Sigma_\mathcal{P} \to \mathbb{Z}^+$ such that $W(\chi)$ is the minimal number of actions that can be taken to drive $\mathcal{P}$ from χ to an accepting state in $F_\mathcal{P}$. If $\chi \in F_\mathcal{P}$, then $W(\chi) = 0$. If $W(\chi) = \infty$, then there does not exist an accepting run originating from χ. Such a state is called a *blocking* state.

The *k-step boundary* about a state χ, denoted $\partial N(\chi, k)$ is the sets of states that can be reached by applying exactly k inputs to the product automaton.

III. Models

A. Robot motion model

We consider a single robot with perfect localization and known dynamics operating in a continuous, partitioned environment. We abstract the motion of the robot to a transition system $Robot = (Q, q_0, Act, Trans, AP, L)$ where Q is a partition of the environment, q_0 is the region in which the agent is initially located, Act is a set of control policies, and $(q,a,q') \in Trans$ if a can drive the robot from q to q'. We assume that given a pair of neighboring regions q, q', the agent can construct a policy a to drive itself from region q to q' if such a policy exists. Tools have been developed for low-level control synthesis in partitioned systems with linear [13] and piecewise affine [23] dynamics. With some conservativism, these can be extended to more realistic dynamics such as the ones modeling unicycles and car-like vehicles [3]. AP is a set of known regional properties of the environment and L is the mapping from a region to the set of properties it satisfies.

Example 1. Consider a robot R_p operating in a grid environment as shown in Figure 1(a) to track a target robot R_t. In this case, $Q = \{1, \ldots 4\}^2 \times \{N, S, E, W\}$ (north, south, east, west) where a state $q = (i, j, dir)$ means that R_p is in grid cell (i, j) and facing direction dir. The set of actions is $Act = \{straight, CW, CCW\}$ which mean go straight, rotate 90° clockwise, and rotate 90° counterclockwise, respectively. $Trans$ is defined according to the implication of each motion and physical limitations of the environment, e.g. $((1, 3, N), straight, (1, 4, N)) \in Trans$, but $(1, 4, N, straight, \cdot) \notin Trans$. $AP = \{\pi_{recharge}, \pi_{data}, \pi_{alarm}, \pi_{reset}, \pi_{obs}\}$ where $\pi_{recharge}$ indicates the presence of a recharging station (green), π_{data} indicates the presence of a station where R_p can transmit data about R_t's position (magenta), π_{alarm} indicates a location in the environment that sounds a proximity alarm (cyan), π_{reset} indicates a shutoff switch for the alarm (blue), and π_{obs} indicates an obstacle (red). A subset of the transition system is shown in Figure 1(b). □

B. Robot sensing model

We associate with the environment a feature that evolves in time synchronously with the robot according to the Markov chain $Env = (S, s^0, P)$. We denote the state of Env at time k as s^k. The initial state s^0 is *a priori* unknown.

At each time k, when the robot moves to state q^k, it measures s^k. The noisy sensor output at time k is a realization $y^k \in R_Y$ of a discrete random variable Y^k. In addition to q^k and s^k, the distribution of Y^k depends on the statistics of the sensor (how well the sensor measures the feature). We denote the conditional measurement distribution as

$$h(y, s, q) = \quad Pr[\text{ measurement is } y \; | Env \text{ in } s, Robot \text{ in } q]. \tag{2}$$

Example 1 (continued). In the above scenario, $S = \{1, \ldots, 4\}^2$ is the set of grid cells in which the tracked

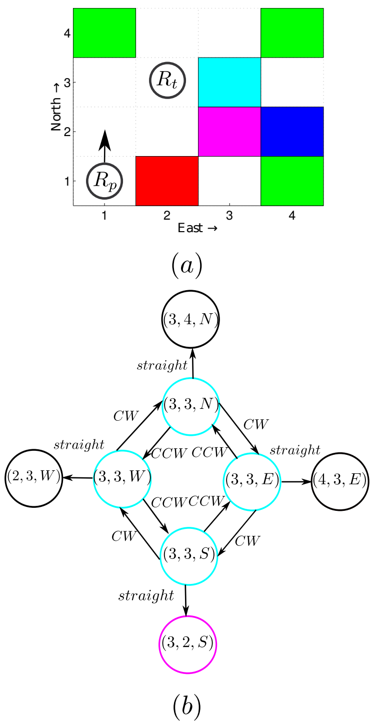

Fig. 1. (a) A robot R_p operates in a grid environment to monitor R_t's position. (b) A subset of the transition system that describes the system in (a).

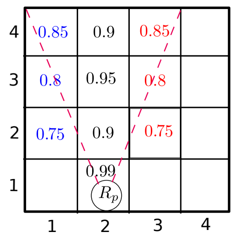

Fig. 2. Measurement model of R_p's forward-facing camera.

agent R_t may be located. s_0 is chosen randomly. P is such that

$$P(s, s') = \begin{cases} p_{move} & s, s' \text{ adjacent} \\ 1 - N_{adj} p_{move} & s' = s \\ 0 & \text{else} \end{cases} \tag{3}$$

where N_{adj} is the number of states that are adjacent to s'. R_p has a noisy, forward-facing camera that can produce measurements in $R_Y = \{0, l, c, r\}$ which correspond to R_t not being observed and R_t being observed in the left, center, or right portion of R_p's field of view denoted by the cone in Figure 2. Part of the measurement likelihood function is summarized in Figure 2. Numbers in blue, black, and red text indicate the probability of l, c, or r being measured by R_p given that R_t is in the indicated cell. □

The robot's estimate of s^k is given via the estimate pmf b^k, called the *belief state* or *belief* , where $b^k(s) = Pr[s^k = s | y^{1:k}, q^{0:k}]$. The initial belief b^0 reflects any *a priori* knowledge about the value of s^0. b^k is maintained via the recursive Bayes filter

$$b^k(s) = \frac{h(y^k, s, q^k) \sum_{s' \in S} P(s', s) b^{k-1}(s')}{\sum_{\sigma \in S} h(y^k, \sigma, q^k) \sum_{s' \in S} P(s', \sigma) b^{k-1}(s')}. \tag{4}$$

The belief b^k evolves over time according to the MDP $Est = (B, b^0, P_{est}, Q)$. Q and b^0 are as defined previously. P_{est} is the probabilistic transition relation such that if b' is the result of applying the Bayes filter (4) with measurement y collected in state q, then $P_{est}(b, q, b')$ is the probability of observing y, i.e.

$$P_{est}(b, q, b') = \sum_{s_1, s_2 \in S^2} h(y, s_2, q) P(s_1, s_2) b(s_1). \tag{5}$$

B is the countably infinite set of all possible beliefs that can be outputs of computing the Bayes filter with initial belief b^0 with an infinite run of $Robot$ along with the resulting measurements. The MDP Est is referred to as the *belief tree* in the partially observable MDP (POMDP) literature [15].

IV. Problem Statement

Our goal is to plan an infinite horizon trajectory $q^{0:\infty}$ for $Robot$ that maximizes the quality of the estimate pmf over time while satisfying a linear temporal logic specification. In [10], [11], we used the expected conditional entropy $E_{Y^{1:t}}[H(b^t)]$ to quantify the expected quality of the estimate that would result from the robot traveling along the finite path $q^{0:t}$. Here, we use a related quantity called the mutual information rate [8], [19]

$$MIR = \lim_{t \to \infty} \frac{I(s^{0:t}; Y^{0:t})}{t}. \tag{6}$$

MIR is the average rate of information gain about the state of Env when a new measurement is taken. We wish to maximize this quantity, as we want to increase the rate at which measurements help to identify the state of Env. Maximizing (6) over the set of actions $Robot$ can take is equivalent to minimizing the *entropy rate*

$$ER = \lim_{t \to \infty} \frac{H(b^t)}{t}. \tag{7}$$

Due to the "infinitely often" acceptance condition of LTL formulae, any trajectory that satisfies an LTL formula can be segmented into a finite length prefix path and an infinite sequence of finite length suffix cycles, which are defined as follows.

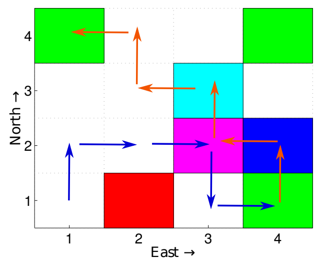

Fig. 3. An example of a prefix path (blue) and a suffix cycle (orange) for the specification (8). The orientation of R_p is not explicitly represented.

Definition 1 (Prefix Path and Suffix Cycle)**.** Let $q^{0:\infty}$ be an infinite horizon trajectory over a transition system TS and ϕ be an LTL formula over the properties AP in TS. Let $\mathcal{P} = TS \times \mathcal{A}_\phi$ and let $\chi^{0:\infty}$ be the infinite run over $\mathcal{P}$ induced by $q^{0:\infty}$. Let l_n be the nth time at which $\chi^{0:\infty}$ intersects F_P. The *prefix path* is the finite sequence $q^{0:l_1}$. A *suffix cycle* is a finite sequence of states $q^{l_n+1:l_{n+1}}$.

Example 1 (continued)**.** The constraints on R_p can be given as the LTL formula

$$\begin{aligned}\phi = \quad & \square \; \diamond \; \pi_{recharge} \wedge \square \; \diamond \; \pi_{data} \wedge \square \; (\neg \pi_{obs}) \\ & \wedge \square \; (\pi_{alarm} \Rightarrow (\neg \pi_{recharge} \; \mathcal{U} \; \pi_{reset})),\end{aligned} \tag{8}$$

which in plain English is "Visit recharging and data upload stations infinitely often while avoiding obstacles. If an alarm is triggered, visit the shutdown switch before visiting the recharging station." Figure 3 shows a prefix cycle and a suffix cycle for ϕ with blue and orange arrows, respectively. □

We wish to minimize the average entropy rate per cycle, given as

$$AERPC = \lim_{n \to \infty} \frac{\sum_{i=0}^{n} \frac{H(b^{l_n}) - H(b^{l_{n-1}})}{l_n - l_{n-1}}}{n} \tag{9}$$

This formulation is similar to the average cost-per-stage problem [9], [4]. With this new objective, we define the constrained persistent monitoring problem as

Problem 1. *Consider an agent that is estimating the state of the environment. Solve*

$$\begin{gathered}\min_{a^0 \ldots \in Act^\infty} E_{Y^{0:l_\infty}}[AERPC] \\ \textit{subject to} \\ \phi \\ l_{n+1} - l_n \leq \ell_{max} \forall n,\end{gathered} \tag{10}$$

where ϕ is an LTL formula and ℓ_{max} is a per-cycle budget.

The budget ℓ_{max} describes energy constraints on the agent's motion, e.g. the maximum amount of time that the agent can be active between recharging.

V. Solution

In [11], we defined a Markov decision process that encapsulated the effect of the robot's action on it's location in the environment, progress towards satisfying the given specification, and it's expected gain in information. Here, we extend this model to LTL specifications and information-gathering over an infinite time horizon.

Definition 2. Full Model MDP
The MDP $FullModel = (\Sigma_{\mathcal{P}} \times B, P_{tot}, Act, (\chi_0, b^0))$, where $\mathcal{P} = Robot \times \mathcal{A}_\phi$ and B, b^0 are derived from Est, describes the synchronous evolution of the robot's position and belief state. The probabilistic transition relationship P_{tot} is defined as

$$P_{tot}((\chi, b), a, (\chi', b')) = P_{est}(b, q', b') I((\chi, a, \chi') \in \Delta_{\mathcal{P}}) I(W(\chi') < \infty) \tag{11}$$

where I is the indicator function ($I(x \in X)$ is 1 if $x \in X$ and 0 if $x \notin X$) and $\chi' = (q', \sigma')$. That is, P_{tot} combines the probabilistic transition relation P with the deterministic transition relation $\Delta_{\mathcal{P}}$ (via the indicator function) while ensuring that no blocking states, i.e. states from which an accepting state cannot be reached, are reachable.

Note that in general B is countably infinite, as starting from an arbitrary belief state b, there is no guarantee that there exists a finite or infinite sequence of actions and observations such that recursively applying the Bayes filter initialized with b will produce b as an output. Thus, the full model MDP cannot be explicitly constructed. In this paper, we only construct finite subsections of the MDP. Given this MDP, we map Problem 1 to the constrained stochastic optimal control problem

$$\begin{gathered}\min_{a^0 \ldots \in Act^\infty} E_{Y^{0:l_\infty}}[AERPC] \\ \text{subject to} \\ l_{n+1} - l_n \leq \ell_{max} \forall n \in \mathbb{N}, \\ a^i \in Act((\chi^i, b^i)) \; \forall i \in \mathbb{N} \\ (\chi^{i+1}, b^{i+1}) \sim P_{tot}(\cdot, a^i, (\chi^i, b^i)) \; \forall i \in \mathbb{N}.\end{gathered} \tag{12}$$

Most methods for solving optimization problems over the belief tree [17], [18] rely on using information about the continuous space of pmfs that can be defined over support S. However, these methods rely on the fact that the cost-to-go function is piecewise linear [20]. Since the entropy rate is highly non-linear, these methods are not applicable.

We propose a hybrid dynamic programming-based method to get around these difficulties. In the low level of our method, we use a receding-horizon algorithm (defined in Section V-A) to choose actions that locally improve the entropy rate and drive the agent to an accepting state in $\mathcal{P}$. Then, at the high level, whenever the agent reaches an accepting state, i.e. completes a suffix cycle, the budget of actions given to the receding horizon algorithm is selected according to a policy that is calculated off-line via value iteration [4] to optimize the infinite-horizon AERPC (Section V-B).

A. Optimization of a single cycle

In [11], we presented a receding horizon algorithm that locally minimized entropy and was guaranteed to satisfy the given scLTL constraint within the budget. Algorithm 1 extends that procedure to handle LTL constraints.

Algorithm 1 Receding horizon planner for completing a single suffix cycle under LTL constraints.

```
function RHP($(\chi, b), \ell, m, n$)
  $k = 0$; $\chi[k] = \chi$;
  while $\chi[k] \notin F_{\mathcal{P}}$ do
    $\Sigma_p$ := PossibleStates($\chi[k]$,$m$,$\ell - k$)
    ($X_p$,$P_{tot}$,$Act_p$) := ConstructMDP($\chi[k]$,$b$,$\Sigma_p$)
    $\mu_l$ :=BellmanIteration($X_p$,$P_{tot}$,$Act_p$)
    if $k \geq \ell - m$ then
      $n := \ell - k$
    for $i := 1$ to $n$ do
      $(\chi[k+1], b)$ := result from applying $\mu(i, (\chi, b))$
      $k++$
  return $(\chi[k], b, k)$
```

At the kth time-step after the invocation of Algorithm 1, the procedure PossibleStates constructs m sets $\Sigma_p[i] \subseteq \Sigma_{\mathcal{P}}$ where Σ_p contains all states $\chi' \in \partial N(\chi[k], i)$ such that $W(\chi') \leq \ell - i - k$, that is, all states reachable from $\chi[k]$ in i actions and from which an accepting state can be reached under the remaining budget. From the sets Σ_p, the procedure ConstructMDP constructs a subset of the full-model MDP such that at the ith level, only pairs (χ, b) such that $\chi \in \Sigma_p[i]$ appear. Next, the algorithm BellmanIteration performs standard Bellman iteration over the constructed MDP where the terminal cost is given as

$$\frac{H(b[m]) - H(b[0])}{m}, \tag{13}$$

where $b[m]$ is the terminal state and $b[0]$ was the initial belief state input to Algorithm 1. After the optimal policy μ_l has been calculated, it is enacted for n time steps. At this point, a new MDP is constructed and a new policy calculated. This process continues until an accepting state is reached.

Applying Algorithm 1 infinitely often provably satisfies the given specification.

Theorem 1. *Let*

$$\ell' = \max_{\chi \in F_{\mathcal{P}}} \min_{\chi' \in \partial N(\chi,1)} W(\chi') + 1.$$

If $\ell' \leq \ell_{max}$, then sequentially applying Algorithm 1 infinitely often with budget at least ℓ' is guaranteed to drive the robot to satisfy the given LTL specification.

Proof. Algorithm 1 is always applied either at the initial state (χ_0, b^0) or at an accepting state (χ, b) such that $\chi \in F_{\mathcal{P}}$. Since we constrain ℓ to be at least ℓ', it is guaranteed that $W(\chi) \leq \ell$. k time steps after the algorithm is applied, the system will be in a state (χ', b') such that $W(\chi') \leq \ell - k$. This means that if $k = \ell$, $W(\chi') = 0$, i.e. $\chi' \in F_{\mathcal{P}}$. So, applying Algorithm 1 one time is guaranteed to drive the system to an accepting state. Applying Algorithm 1 infinitely often guarantees that the system will be in an accepting state infinitely often, thus satisfying the Buchi acceptance condition and therefore satisfying the given LTL specification. □

Further, a single application of Algorithm 1 is tractable.

Proposition 1. *The time complexity of Algorithm 1 with budget ℓ, planning horizon m, and action horizon n is $O(|Act|^m |R_Y|^m |S| \lceil \frac{\ell}{n} \rceil)$.*

B. Choosing cycle budgets

Although applying Algorithm 1 infinitely often with a fixed budget ℓ is guaranteed to satisfy the LTL specification, there is no guarantee that the local information gathering performs well over an infinite time horizon. We propose to pair the local information gathering with long-term planning by finding a policy $\mu_g : F_{\mathcal{P}} \cup \{\chi_0\} \times B \rightarrow \mathbb{N}$ that maps an initial or accepting state in the automaton and a belief state, e.g. the configuration of the robot at the end of a cycle, to the budget ℓ that should be given to the receding horizon planner in the next cycle. The hierarchal structure of the algorithm is illustrated in Figure 4.

It may seem that the longer the budget ℓ handed to the receding horizon planner, the better we expect the performance to be. As ℓ increases, on average more states are

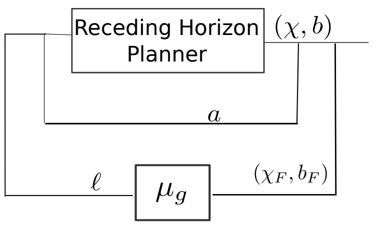

Fig. 4. A block diagram illustrating the hierarchal dynamic programming algorithm developed in this section. The receding horizon planner applies Algorithm 1 to determine the next action to take based on the current automaton state and belief state pair (χ, b) and the budget of actions ℓ between satisfactions. When the system reaches an accepting configuration, denoted (χ_F, b_F), the policy μ_g is called to determine the budget ℓ that should be used in the next round of receding horizon planning.

included in the MDPs constructed by Algorithm 1. However, the agent can take an action that locally performs better but which may cause the long-term performance to deviate far from optimal behavior. If this action were not present, the deviation would not occur. We address this by characterizing the statistical performance of the receding horizon algorithm for different budgets and use this information to construct the optimal policy μ_g. The policy construction is performed off-line before the agent is deployed.

Algorithm 2 details how μ_g is calculated. We use simulation to characterize the per-cycle performance of the receding horizon algorithm. We consider a finite set of states $T \subset F_{\mathcal{P}} \times B$ in our optimization. Algorithm 2 begins by performing j_{max} simulations of Algorithm 1 from the initial state (χ^0, b^0) for each value of ℓ from $\min_{\chi'' \in \partial N(\chi^0,1)} W(\chi'') + 1$ to ℓ_{max}. The result of each simulation is a state (χ', b') that is reachable from the initial state. Note that in our algorithm, we group similar belief states together. If two states are ϵ close in the 1-norm, we consider them identical. This is to help mitigate the state explosion problem and to prevent unnecessary calculation of similar quantities. The cost of going from (χ, b) to (χ', b') under budget ℓ, denoted $g((\chi, b), \ell, (\chi', b')) = \frac{H(b') - H(b)}{\ell}$, is calculated and recorded. Similarly, the frequency of transitioning from state (χ, b) to (χ', b'), under budget ℓ, denoted $P((\chi, b), \ell, (\chi', b'))$, is calculated from the set of simulations and recorded. The reachable state (χ', b') is then added to the set the set of states still to be checked C. After this set of simulations is completed, the states of C are iterated through and more simulations occur.

T, P, and g are populated until $|T| \geq N$ and all of the accepting states in $F_{\mathcal{P}}$ have been visited or there are no states left to be checked (due to all of the states found being ϵ-close to members of T). At this point, we invoke a procedure to make the MDP described by P,g, and T recurrent. That is, for every state $(\chi, b) \in T$ for which P is undefined, we apply Algorithm 1 for each possible value of ℓ. Then, we replace the resulting state (χ', b') with (χ_c, b_c) the closest state already enumerated. The resulting costs $g((\chi, b), \ell, (\chi_c, b_c))$ and transition probabilities $P((\chi, b), \ell, (\chi_c, b_c))$ are recorded. In short, this procedure ensures that every state in T is connected via P to another state in T. Finally, Algorithm 2 uses value iteration to find the optimal policy

$$\begin{aligned}\mu_g(\chi, b) = \quad & \arg\min_{\ell} \textstyle\sum_{(\chi',b')\in T} P((\chi, b), \ell(\chi', b')) \\ & [g((\chi, b), \ell(\chi', b') + J^\infty((\chi', b'))],\end{aligned} \tag{14}$$

for all $(\chi, b) \in T$ where $J^\infty : T \to \mathbb{R}$ is the infinite-horizon cost-to-go function. [4] Note that this optimization does not directly minimize the average entropy rate per cycle. Rather, it minimizes the sum of the expected entropy rates per cycle. However, since we are minimizing the entropy rate on a cycle-by-cycle basis, the average rate will also be minimized.

Algorithm 2 Constructs the policy mapping current state to optimal budget.

```
1: function OptimalBudget($\chi_0$,$b^0$,$\ell_{max}$,$m$,$n$)
2: $T = \{(\chi_0, b^0)\}$; $C = \{(\chi_0, b_0)\}$; $Acc = F_{\mathcal{P}}$
3: while $|T| \leq N$ do
4:     $(\chi, b)$ = $C$.pop()
5:     for $\ell = \min_{\chi'' \in \partial Nin} W(\chi'') + 1$ to $\ell_{max}$ do
6:         for $j = 1$ to $j_{max}$ do
7:             $(\chi', b', k)$ = RHP($(\chi, b), \ell, m, n$)
8:             if $\exists(\chi', b'') \in T$ s.t. $||b'' - b'||_1 < \epsilon$ then
9:                 $b' = b''$
10:            else
11:                $T$.add($(\chi', b')$); $C$.add($(\chi', b')$)
12:            $g((\chi, b), \ell, (\chi', b'))$ = $\frac{H(b')-H(b)}{k}$
13:            $P((\chi, b), \ell, (\chi', b'))$ += $\frac{1}{j_{max}}$
14:            $Acc = Acc \setminus \{\chi'\}$
15:    if $C = \emptyset$ then
16:        Break;
17:    if $|T| = N$ and $Acc \neq \emptyset$ then
18:        $N = N + 1$
19: $(P, g, T)$ = MakeRecurrent($P, g, T$)
20: $\mu_g$ = ValueIteration($P, g, T$)
21: return $\mu_g, T$
```

Algorithm 3 summarizes our approach. We calculate the optimal budget policy off-line. Then, we use the policy on-line when applying Algorithm 1 infinitely often. If a state is reached that is not in the pre-calculated set T, we apply the mapping μ_g to the closest state contained in T.

Algorithm 3 Solution to constrained persistent monitoring

```
function PersistentMonitor($\chi_0$,$b^0$,$\ell_{max}$,$m$,$n$)
$\mu_g, T$ = OptimalBudget($\chi_0$,$b^0$,$\ell_{max}$,$m$,$n$);
$\chi = \chi_0$; $b = b^0$;
while TRUE do
    if $(\chi, b) \notin T$ then
        $b = \arg\min_{b' \in T} ||b - b'||_1$
    $(\chi, b, k)$ =RHP($(\chi, b), \mu_g((\chi, b)), m, n$)
```

VI. Case Study

We implemented Algorithm 3 in software and applied it to a simulation of the scenario described in the running example Example 1.

The probability that the tracked target R_t moves to an adjacent cell is $p_{move} = 0.15$. The optimal policy μ_g was computed with parameters $\epsilon = 0.01$, $N = 1500$, $j_{max} = 100$, $\ell_{max} = 15$, $m = 2$, and $n = 1$. The computation required approximately 6 hrs. of processor time. We performed 500 Monte Carlo trials of the system in which the system was simulated until 5 suffix cycles were completed, i.e. until R_p had visited a recharging station and a data upload station 5 times. This was compared to 500 "random walk" simulations in which at every time $k+l_n$, the agent selected its action uniformly at random from the set $\{a \in Act|(q^{k+l_n}, a, q') \in Trans, W(q') \leq \ell_{max} - k\}$. This random walk policy is also guaranteed to satisfy the constraints on the system. The results from the simulations are summarized in the histograms in Figure 5. The random walk policy resulted in an average entropy rate of 0.0420 bits/action, while the average entropy rate when using Algorithm 3 was -0.0090 bits/action. A two-sample t-test confirmed that this difference in means is statistically significant with p-value less than 10^{-95}. Further, 3.5 % of the random walk trials resulted in negative entropy rates while 64.8% of the trials with Algorithm 3 had negative rates, i.e. on average gain information about the location of R_t with each action taken. This indicates that in addition to having better mean performance, Algorithm 3 has better performance more often.

VII. Conclusions

In this work, we have extended our previous results in constrained informative planning to handle infinite-horizon constraints modeled as LTL formulae. Our procedure exploits an on-line, receding horizon planner that

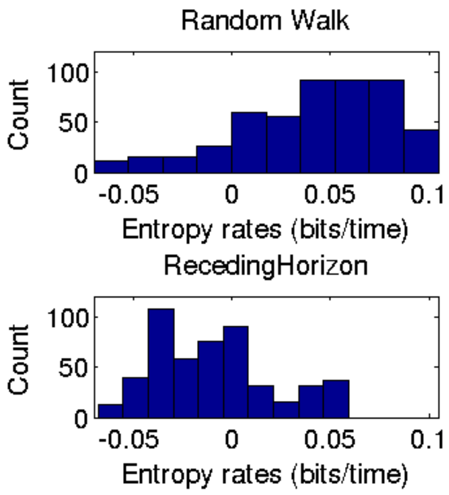

Fig. 5. (a) random walk policy (b) Algorithm 3.

is guaranteed to satisfy the given constraints. In order to mitigate the myopia that is inherent with local receding horizon planners, we developed and implemented an off-line algorithm which is used to select the number of allowable actions that the receding horizon planner can take between subsequent cycles of the robot's trajectory. We evaluated our procedure in simulation and the results indicate that the receding horizon implementation significantly outperforms a random walk simulation.

In the future, we will investigate more sophisticated reinforcement learning algorithms [12] to mitigate the computational complexity of our high-level planner. That is, we will search for ways to limit the number of states considered in the high-level planner to only those that are most representative or perform the best. We intend to extend these results to multi-agent systems by combining tools to distribute temporal logic requirements among agents [7] and results from multi-agent persistent monitoring.

References

[1] Ebru Aydin Gol, Mircea Lazar, and Calin Belta. Language-guided controller synthesis for discrete-time linear systems. In *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, pages 95–104, New York, NY, USA, 2012.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[3] C. Belta, V. Isler, and G. J. Pappas. Discrete abstractions for robot planning and control in polygonal environments. *IEEE Trans. on Robotics*, 21(5):864–874, 2005.

[4] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000.

[5] A. Bhatia, L.E. Kavraki, and M.Y. Vardi. Motion planning with hybrid dynamics and temporal goals. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 1108 –1115, Dec. 2010.

[6] Christos G. Cassandras and Xuchao Lin. Optimal control of multi-agent persistent monitoring systems with performance constraints. In Danielle C. Tarraf, editor, *Control of Cyber-Physical Systems*, volume 449 of *Lecture Notes in Control and Information Sciences*, pages 281–299. Springer International Publishing, 2013.

[7] Yushan Chen, Xu Chu Ding, A. Stefanescu, and C. Belta. Formal approach to the deployment of distributed robotic teams. *Robotics, IEEE Transactions on*, 28(1):158 –171, feb. 2012.

[8] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2nd edition, 2006.

[9] Xu Chu Ding, C. Belta, and C.G. Cassandras. Receding horizon surveillance with temporal logic specifications. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 256 –261, Dec. 2010.

[10] Austin Jones, Mac Schwager, and Calin Belta. A receding horizon algorithm for informative path planning with temporal logic constraints. In *International Conference on Robotics and Automation (ICRA)*, 2013.

[11] Austin Jones, Mac Schwager, and Calin Belta. Formal synthesis of optimal information-gathering policies. *IEEE Trans. on Robotics*, Submitted.

[12] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

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

[14] Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 2001. volume 19, pages 291-314.

[15] Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*, 2010.

[16] Xiaodong Lan and M. Schwager. Planning periodic persistent monitoring trajectories for sensing robots in gaussian random fields. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2415–2420, May 2013.

[17] Joelle Pineau. *Tractable Planning Under Uncertainty: Exploiting Structure*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2004.

[18] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, pages 1–51, 2012.

[19] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656, 1948.

[20] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088, 1973.

[21] S. L. Smith, M. Schwager, and D. Rus. Persistent robotic tasks: Monitoring and sweeping in changing environments. *IEEE Transactions on Robotics*, 28(2):410–426, April 2012.

[22] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata, volume 1043 of Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996.

[23] Boyan Yordanov, Jana Tumova, Ivana Cerna, Jiri Barnat, and Calin Belta. Temporal logic control of discrete-time piece-wise affine systems. *IEEE Transactions on Automatic Control*, 57:1491–1504, 2012.