Formal synthesis of optimal information-gathering policies

Austin Jones, Mac Schwager, Calin Belta

Division of Systems Engineering, Boston University, Boston, MA

This paper considers the problem of informative path planning under temporal logic (TL) correctness constraints. For example, a robot deployed to a building after a natural disaster must explore the area and report possible locations of survivors with minimum uncertainty. The robot must satisfy the correctness requirement "Always avoid obstacles. Report data to rescuers at a data upload hub before exiting the scene. If a location with suspected fire is visited, investigate and report immediately." In this work, we map the constrained informative path planning problem to a stochastic optimal control problem over a Markov decision process that integrates a robot's motion under a given TL specification with its sensing model. We develop an optimal dynamic programming algorithm and a receding horizon approximation with significantly reduced computational cost. Both solutions are guaranteed to satisfy the given specification and are evaluated using simulations and experiments with ground robots. Our results show that the receding horizon solution approximates the optimal method closely, which indicates usefulness for persistent information gathering tasks.

I. Introduction

We address the problem of constructing a control policy for a mobile agent that both satisfies a high-level mission with respect to its environment given as a temporal logic (TL) formula and collects the maximum amount of information about an *a priori* unknown feature. Consider a robot deployed to a building after a natural disaster to locate survivors. Its tour must conform to requirements such as avoiding collisions with obstacles, visiting locations where it can transmit data to rescuers after visiting areas of interest, and exiting the building at one of several possible exits either at the end of its tour or if it becomes damaged while in operation. These requirements can be expressed naturally as a TL formula. In addition, the robot must ensure it locates survivors as precisely as possible. Informative path planning deals with the problem of planning trajectories such that the agents' sensors take the best possible measurements, usually by optimizing over some information-theoretic quantity like mutual or Fischer information. In this paper, we consider the problems of TL planning and informative path planning simultaneously. We provide dynamic programming algorithms that are guaranteed to satisfy complex motion constraints while also maximizing the informativeness of the resulting path. The performance of these algorithms are characterized with theoretical results, Monte Carlo simulations, and experiments with a ground robot localizing a randomly moving agent.

The agent's estimate of the feature is given as a probability mass function (pmf) over a discrete set that is maintained over time via recursive Bayesian estimation. Paths are optimized with respect to the Shannon entropy of this pmf [30].

We consider constraints on the motion of the agent that can be described with syntactically co-safe linear temporal logic (scLTL) [20], [23], a fragment of the more familiar LTL [3] suitable for describing finite time properties. These formulae can be used to express a large class of natural constraints on the motion of robots, such as "Eventually reach the target region while avoiding unsafe regions. Visit region A or B before going to the target region. If you enter region C, go immediately to region D."

The key observation that enables our main results is that it is possible to construct a Markov decision process (MDP) that simultaneously encapsulates the robot's motion, its estimation process, and its progress towards satisfying the given scLTL formula. The problem under consideration can thus be framed as a constrained stochastic optimal control problem that can be solved via stochastic dynamic programming (SDP). Two methods are developed: an off-line implementation guaranteed to return the optimal policy of a given length, and an on-line, receding-horizon approximation with lower computational cost. All generated policies satisfy the scLTL constraints on the robot's motion.

Preliminary results appeared in conference proceedings [13], in which we showed how to combine results from formal methods and information theory to model the TL-constrained informative path planning problem. The main result from [13] is a pair of algorithms, an off-line entropy minimization and on-line receding horizon algorithm, that were both shown to satisfy the given

temporal logic mission. The optimization in [13] was performed over the set of possible paths that could satisfy the given mission. Here, we optimize over the set of possible feedback control policies that can drive the robot to satisfy the mission. This produces a reactive solution; the dynamic programming algorithms presented here in expectation outperform the previously presented results (See Section V). In this paper, we assume the underlying feature evolves probabilistically, while in our previous paper we assumed a static feature. We also include new experimental and simulation results to verify our solution.

II. Related Work

Informative path planning is the classical sensor placement problem, i.e. placing a collection of sensors in an environment such that the maximum amount of information about a sensed variable is gained [1], with mobile sensors. Previous works have focused on optimizing information-theoretic measures. Many sophisticated algorithms exist for performing informative path planning in discretized environments for single or multiple agents using e.g. recursive greedy planning [31] or sequential dynamic programming [25]. Other constraints on the agents' motion such as communication constraints [12], [16], collision avoidance [7], [8], [11], and environmental hazards [29] has also been considered. Our problem introduces complex motion constraints given as scLTL formulae.

One of the algorithms presented in this work relies on a receding-horizon implementation to reduce computational burden. Receding horizon planning grew out of classical model-predictive control [24], [27], [28], a family of control algorithms in which the control policy is calculated based on a prediction of the future behavior of the system given current and prior knowledge of the system's state and parameters. There exist receding horizon algorithms that are used to maximize local reward gathering while guaranteeing satisfaction of TL constraints [10], [37]. In contrast to these two works, our models involve optimizing a stochastic objective function.

Temporal logics [3] have long been used as formal description languages. Connections of temporal logic to automata [34] enable procedures for synthesizing decision policies over labeled finite models that can provably satisfy TL specifications. In addition, control system abstractions can be constructed to allow synthesis of control policies for systems with linear [17], piecewise affine [38], or polynomial [32] dynamics. These methods have seen wide usage for motion planning in robotics, as TL naturally expresses many high-level robotics missions. Sophisticated TL-based motion planning algorithms have been developed to compensate for uncertainty in the dynamics [22], sensors [18], or environment [36], react to changes [19], and take advantage of sample-based motion planning [15], [35]. Here we consider TL-based motion planning with the added objective of collecting information about the robot's environment using noisy sensors. Motion planning algorithms for the particular logic, scLTL, used in this work also exist [2], [33].

III. 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 of states $x^1 \dots x^t$ where x^i is the value of x at time i. The set of all finite and set of all infinite words over alphabet Σ are denoted by Σ^* and Σ^∞, respectively. For a discrete random variable X, we use R_X, p_X, and $E[X]$ to denote its range-space, probability mass function (pmf), and expectation, respectively. If Y is a function of random variable X, we denote the expectation of Y with respect to X as $E_X[Y]$. We denote the *conditional entropy* between random variables X and Y [9] as $H(X|Y) = H(p_X|p_Y) = -\sum_y \sum_x p_{X,Y}(x,y) \log(p_{X|Y}(x|y))$, where $p_{X,Y}$ is the joint distribution of X and Y and $p_{X|Y}$ is the distribution of X given Y.

A. Models

A *(deterministic) transition system* (TS) [3] 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 deterministic 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 finite *run* of a weighted transition system is a sequence of states $q^0 q^1 \dots \in Q^*$ such that $q^0 = q_0$, and $\exists a^i \in Act$ such that $(q^i, a^i, q^{i+1}) \in Trans$ $\forall i = 0, 1, \dots$. An *output trace* of a run $q^0 q^1 \dots$ is a word $w = w^0 w^1 \dots$ where $w^i = L(q^i)$.

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

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 $s^0 s^1 \dots s^\ell$ with $P(s^i, a, s^{i+1}) > 0$ for some $a \in Act(s^i)\ \forall i = 0, \dots, \ell$.

In this paper, we assume that the reader is familiar with stochastic optimal control over MDPs, in which an accumulated cost defined with respect to sample paths of an MDP is minimized, and its solution via dynamic programming. [6]. A *policy* $\mu : \mathbb{N} \times S \to Act$ maps a time and state of the MDP such that $\mu(k,s)$ is the action to be taken at time k in state s. In this work, we denote the optimal policy obtained through Bellman iteration (a dynamic programming algorithm) as μ^*.

B. Temporal logics and automata

An scLTL formula over a set AP is inductively defined as follows [20]:

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

where $p \in AP$ and ϕ is an scLTL formula. The logical operators $\vee, \wedge$, and $\neg$ are disjunction, conjunction, and negation, respectively, and the temporal operators $\mathcal{U}, \bigcirc$, and $\Diamond$ are until, next, and eventually, respectively. We also use Boolean implication $\Rightarrow$, where $(\phi_1 \Rightarrow \phi_2) = (\neg\phi_1 \vee \phi_2)$. scLTL is defined over words $w = w^0 w^1 \dots \in (2^{AP})^*$. The notation $w \models \phi$ is used to mean that w *satisfies* an scLTL formula ϕ. The set of all words that can satisfy a formula ϕ is called the *language* of ϕ, denoted $\mathcal{L}(\phi)$.

In this work, we define constraints on the motion of a robot as scLTL formulae over labels of possible locations. Properties that can be described by scLTL include

- reachability ($\Diamond\ p_{\text{goal}}$, eventually go to p_{goal}),
- finite-time avoidance ($\neg p_{\text{avoid}} \mathcal{U} p_{\text{goal}}$, avoid p_{avoid} before reaching p_{goal}),
- goal sequencing ($\neg p_{\text{goal}_2} \mathcal{U} p_{\text{goal}_1} \wedge \Diamond\ p_{\text{goal}_2}$, go to p_{goal_1} and then p_{goal_2}), and
- reactivity ($p_{\text{trigger}} \Rightarrow \Diamond\ p_{\text{effect}}$, if p_{trigger} is seen, go to p_{effect}).

A *(deterministic) finite state automaton* (FSA) is a tuple $FSA = (\Sigma, \Pi, \Sigma_0, F, \Delta_{FSA})$ where Σ is a finite set of states, Π is an input alphabet, $\Sigma_0 \subseteq \Sigma$ is a set of initial states, $F \subseteq \Sigma$ is a set of final (accepting) states, and $\Delta_{FSA} \subseteq \Sigma \times \Pi \times \Sigma$ is a deterministic transition relation. An *accepting run* r_{FSA} of an automaton FSA on a finite word $\pi^0 \pi^1 \dots \pi^j \in \Pi^*$ is a sequence of states $\sigma^0 \sigma^1 \dots \sigma^{j+1}$ such that $\sigma^{j+1} \in F$ and $(\sigma^i, \pi^i, \sigma^{i+1}) \in \Delta_{FSA}\ \forall i \in [0, j]$. We call the set of words that can lead to an accepting run on FSA the *language* of FSA, denoted $\mathcal{L}(FSA)$.

Given an scLTL formula ϕ over the set of atomic propositions AP, there exist algorithms for creating an FSA with input alphabet 2^{AP} that accepts all and only words satisfying ϕ, i.e. an automaton FSA_ϕ such that $\mathcal{L}(FSA_\phi) = \mathcal{L}(\phi)$.

The *product automaton* between a deterministic transition system $TS = (Q, q_0, Act, Trans, AP, L)$ and an FSA $FSA_\phi = (\Sigma, 2^{AP}, \Sigma_0, F, \Delta_{FSA})$ is an FSA $\mathcal{P}_\phi = TS \times FSA_\phi = (\Sigma_\mathcal{P}, \chi^0, Act, F_\mathcal{P}, \Delta_\mathcal{P})$ [3]. $\Sigma_\mathcal{P} \subseteq Q \times \Sigma$ 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 $\Delta_\mathcal{P} = \{(q,\sigma), p, (q',\sigma') | (q,p,q') \in Trans, (\sigma, L(q), \sigma') \in \Delta_{FSA}\}$. The state of the automaton at time k, (q^k, σ^k) is denoted as χ^k for short.

We define the *distance to acceptance* 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 $\mathcal{P}_\phi$ 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 doesn't exist an accepting run originating from χ.

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.

IV. Problem formulation

In this section, we present the agent's motion and sensing model and formalize the scLTL-constrained informative path-planning problem as a discrete optimization.

A. Motion model

We consider a single robot moving on a graph-like environment described by a transition system $Robot = (Q, q_0, Act, Trans, AP, L)$. Q is a finite set of states (the nodes of the graph) and q_0 is the robot's initial state. Act is a set of actions that the robot can enact. $Trans$ is a transition relation (set of edges in the graph) such that $(q, a, q') \in Trans$ if action a drives $Robot$ from state q to q'. AP is a set of atomic propositions (properties) and $L : Q \to 2^{AP}$ is the mapping from a state to the set of propositions it satisfies. We define a discrete clock k that is initialized to zero and increases by 1 each time $Robot$ takes an action. We denote the $Robot$'s state at time k as q^k.

Remark 1. A transition system such the one described above can be easily constructed by partitioning a planar or 3D environment. The states would correspond to the regions in the partition. The transitions and the corresponding actions would capture adjacency relations and

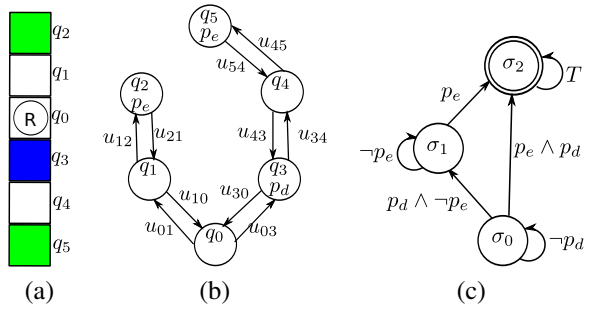


Fig. 1. Illustration of Example 1. (a) The hallway environment in which the robot (denoted by R) operates. The data upload regions are blue and the exits are green. The labels q_i for the regions correspond to states in the transition system $Robot$. (b) Transition system constructed from the hallway environment. Action u_{ij} corresponds to a control driving the robot from region q_i to q_j. The labels p_d and p_e correspond to data upload and exit regions, respectively. (c) FSA constructed from scLTL formula ϕ_{ex} given in (5). Accepting states are indicated with double circles.

feedback controllers driving all states from a region to another, respectively. Such controllers can be efficiently constructed for affine / multi-affine dynamics and simplicial / rectangular partitions [4], [5]. Such techniques can be extended to more complicated systems, such as unicycle and car-like dynamics through the use of input-output linearization techniques.

See Figure 1(a)-(b) for an example of a transition system corresponding to Example 1, a scenario that will serve as a running example throughout this paper.

Example 1. A ground robot equipped with a noisy camera is deployed to an office building after a natural disaster to locate survivors as precisely as possible. We consider a problem in which the robot operates in a hallway divided into six regions (Figure 1(a)). The robot must visit a region where it can upload data to rescue workers before eventually exiting the hallway. □

B. Sensing model

We associate with the environment a feature that evolves in time synchronously with the clock k 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 using noisy sensors. The 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 time-invariant conditional measurement distribution as

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

Example 1 (continued)**.** Let $S = \{0, 1\}^6$. The jth element of a state $s \in S$ corresponds to whether or not a survivor is located in region q_{j-1}, e.g. $s^k = [0, 1, 0, 0, 0, 0]$ means at time k a survivor is located in region q_1 and no other regions contain survivors (See Figure 1(a)). s^0 represents the initial locations of survivors, and P represents the probability that survivors move between neighboring regions, e.g. $P([0, 1, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0])$ is the probability that a survivor moves from region q_1 to q_0. For simplicity, we assume that at most one survivor may be located in a region. The robot uses a camera to survey its surroundings and runs a detection algorithm on the gathered frames to estimate whether or not a survivor is located in its current hallway region, i.e. $y^k = 1$ if the algorithm produces a detection and $y^k = 0$ if it does not. □

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 belief state is initialized with a pmf b^0 that reflects the *a priori* belief about the value of s^0 and maintained via the Bayes filter

$$b^k(s) = \frac{h(y^k, s, q^k) \sum_{s' \in S} P(s', s) b^{k-1}(s')}{\sum_{\sigma \in S} h(y^k, \sigma, q^k) \sum_{s' \in S} P(s', \sigma) b^{k-1}(s')}. \quad (3)$$

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 (3) with measurement y collected in state q, then $P_{est}(b, q, b')$ is the total probability of observing y, i.e.

$$P_{est}(b, q, b') = \sum_{s_1, s_2 \in S^2} h(y, s_2, q) P(s_1, s_2) b(s_1), \quad (4)$$

where a was the action that drove $Robot$ to q. 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 a run of $Robot$ along with the measurements the robot takes at each state in the run.

In general, the states and transitions of Est can be arranged in a tree structure with root b^0. The children of a node b in the tree are $\{b' | \exists q \in Q, P_{est}(b, q, b') > 0\}$. Because of this branching, Est is also referred to as the *belief tree* in the partially observable MDP (POMDP) literature [21].

C. Constrained maximally informative path planning

We are interested in planning a finite trajectory for $Robot$ such that the uncertainty about the state of Env is minimized when $Robot$ completes the mission given by an scLTL formula ϕ over the labeled regions of the environment.

Example 1 (continued)**.** The regions where data can be uploaded are labeled with p_d and the regions with exits are labeled with p_e. The constraints on the motion of $Robot$ may be expressed as

$$\phi_{ex} = \diamondsuit\, p_e \wedge (\neg p_e \mathcal{U} p_d), \tag{5}$$

meaning "Go to a data upload region (p_d) before going to an exit (p_e)." The corresponding finite state automaton $FSA_{\phi_{ex}}$ is shown in Figure 1 (c). □

We quantify the uncertainty in a belief b with its Shannon entropy $H(b)$. Our goal is to select actions $a^{0:t-1}$, $a^i \in Act\ \forall i \in 1, \ldots, t$, such that in expectation $H(b^t)$ is minimized. We predict the expected entropy of the belief that results from following a given path $q^{0:t}$ by calculating the conditional entropy $H(b^t|b^0, Y^{0:t}, q^{0:t})$ for short.

We allow $Robot$ to take at most ℓ actions. This budget constraint reflects energy limitations on the robot. Each action consumes energy, and the robot should measure its environment and complete its mission (satisfy ϕ) before it drains its reserves.

The scLTL-constrained informative path planning problem is formulated in Problem 1.

Problem 1 (scLTL-constrained informative path planning)**.** Given a robot with model $Robot$ operating in an environment Env, a finite budget ℓ, and an scLTL formula (mission) ϕ over AP, solve

$$\begin{gathered} \min_{a^{0:t-1}} E_{Y^{0:t}}[H(b^t|b^0, Y^{0:t}, q^{0:t})] \\ \text{subject to} \\ t \leq \ell \\ \phi \text{ is satisfied} \end{gathered} \tag{6}$$

V. Dynamic Programming Solutions

In this section, we connect Problem 1 to Markov decision processes (MDPs) and present two solutions: an off-line dynamic programming approach and an on-line receding-horizon dynamic programming approach. Analysis of the complexity and quality of each algorithm is given. All proofs are given in the Appendix.

For a given scLTL formula ϕ, let $\mathcal{P}_\phi = Robot \times FSA_\phi$ as illustrated in Figure 2. Since the motion of an agent is deterministic, the correctness of a run of $Robot$ with respect to ϕ can be determined precisely via $\mathcal{P}_\phi$, but its estimate b evolves probabilistically according to the MDP Est. As $Robot$ and Est evolve synchronously, we can combine them into a single MDP that encapsulates the robot's movement and estimation process.

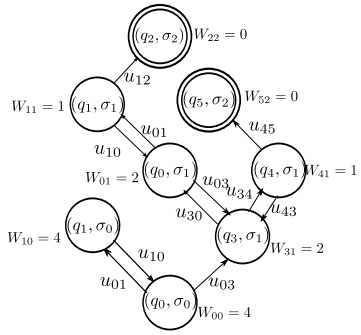


Fig. 2. Product automaton constructed from the scenario in Example 1. Each state (q_i, σ_j) is annotated with its distance to acceptance, $W_{ij} = W((q_i, \sigma_j))$. Accepting states are denoted with double circles.

Definition 1. Full Model MDP

Consider a robot whose motion is modeled by $Robot$ (Section IV-A) under a temporal logic constraint ϕ. Simultaneously, the robot is estimating the state of Env through the process Est (Section IV-B). The MDP $FullModel = (\Sigma_{\mathcal{P}_\phi} \times B, P_{tot}, Act, (\chi_0, b^0))$ describes the synchronous evolution of the robot's position and estimate pmf. The probabilistic transition relationship P_{tot} is defined as

$$\begin{aligned} P_{tot}((\chi, b), a, (\chi', b')) = \\ P_{est}(b, q', b') I((\chi, a, \chi') \in \Delta_{\mathcal{P}_\phi}) \end{aligned} \tag{7}$$

where I is the indicator function ($I(x \in X)$ is 1 if $x \in X$ and 0 if $x \notin X$) and $\chi' = (q', \sigma')$.

$FullModel$ has a tree structure with root (χ_0, b^0). The states at the $i+1$st level are the set of states that are reachable from taking one action and making one observation from any state in the ith level.

Example 1 (continued)**.** Figure 3 shows the full model MDP constructed from the hallway scenario shown in Figure 1. To simplify the visual representation, edges in the MDP are grouped together via action, and the probabilistic transition relation is not explicitly shown. Note that any sequence of actions in this MDP will result in a state in which the automaton state component is accepting (denoted by double circles). The subscripts of the belief states correspond to states in the MDP Est (not shown) which were indexed sequentially as the tree was constructed, i.e. the subscript index increased by 1

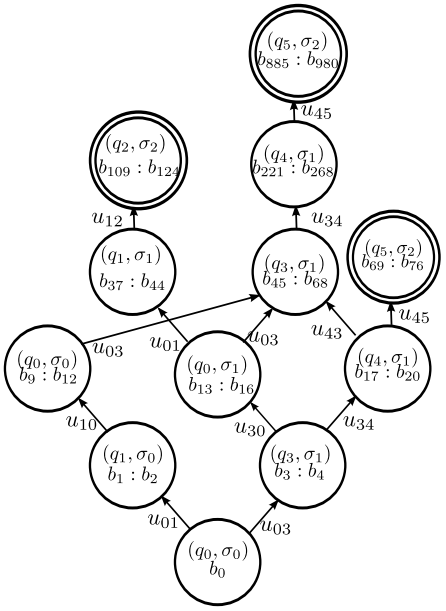


Fig. 3. Full model MDP for hallway scenario (Example 1 with planning budget $\ell = 5$.

each time a state was added to the topmost level of the tree. The temporal logic constraint (5) limits branching. This MDP has 218 total states, compared to 980 states in Est with depth 5. □

Remark 2. In our algorithms, we only consider a finite subset of $FullModel$, namely the states and transitions such that every sample path of length at most ℓ beginning from the initial state (χ_0, b^0) is guaranteed to satisfy ϕ. The number of states in this finite subset of $FullModel$ is less than or equal to the number of states in the first $\ell+1$ levels of Est. This is due to the fact that the children of a node in Est correspond to all actions that can be taken in the current state. However, taking certain actions may violate ϕ or ensure that it cannot be satisfied within the remaining budget. The children of such actions do not appear in $FullModel$.

A. Off-line dynamic programming

We can cast Problem 1 as a constrained stochastic optimal control problem defined over $FullModel$.

Proposition 1. *Let $FullModel$ be defined according to Definition 1. Then, Problem 1 is equivalent to the following constrained stochastic optimal control problem:*

$$\min_{a^{0:t-1}} E_{Y^{0:t}}[H(b^t)] \quad \textit{subject to} \tag{8a}$$

$$t \leq \ell \tag{8b}$$

$$a^i \in Act((\chi^i, b^i)) \ \forall i = 0, \ldots, t-1 \tag{8c}$$

$$(\chi^{i+1}, b^{i+1}) \sim P_{tot}(\cdot, a^i, (\chi^i, b^i)) \tag{8d}$$

$$\chi^t \in F_{\mathcal{P}_\phi}. \tag{8e}$$

The off-line, dynamic programming solution to Problem 1 is outlined in Algorithm 1 [1].

Algorithm 1 Returns a policy for ℓ time units that satisfies ϕ and minimizes $E_{Y^{0:t}} H(b^t)$

```
function FiniteHorizonDP(P_φ,χ^0,b^0,ℓ)
if W(χ^0) > ℓ then
    return None
automatonStates := PossibleStates(P_φ,χ^0,ℓ, ℓ)
(MDPStates,P_tot,actionSet) := ConstructMDP(P_φ, χ^0,b^0,automatonStates,ℓ)
return BellmanIteration(MDPStates,P_tot,actionSet)
```

First, the algorithm ensures that the planning budget ℓ is long enough to allow $Robot$ to satisfy ϕ, i.e. checks whether the distance to accepting $W(\chi^0)$ is at most ℓ. Next, Algorithm 2 calculates ℓ subsets of $\Sigma_{\mathcal{P}_\phi}$ automatonStates[i], $i = 1, \ldots, \ell$. Each state $\chi \in$ automatonStates[i] is reachable from χ^0 in exactly i transitions (is in $\partial N(\chi^0, i)$) and can reach an accepting state within the remaining budget of $\ell - i$ transitions ($W(\chi) \leq \ell - i$).[2]

Algorithm 2 Calculate all product automaton states that can lead to accepting runs in ℓ or fewer transitions.

```
function PossibleStates(P_φ,χ^0,m,ℓ)
automatonStates[0] := χ^0
for i := 1 to m do
    automatonStates[i] = {χ ∈ Σ_{P_φ} | W(χ) ≤ ℓ − i} ∩ ∂N(χ^0, i))
return automatonStates
```

Next, we use Algorithm 3 to build a finite subset of the infinite-dimensional MDP $FullModel$ such that

[1]The cost in (8) is defined with respect to a pmf, which may recall stochastic optimal control of POMDPs [14]. It is tempting to use established approximation methods such as point-based value iteration [26] to optimize over the continuous space of pmfs rather than enumerate a finite number of states. However, in our formulation, the cost of a pmf $H(b)$ is highly non-linear in the components $\{b(s)|s \in S\}$. Since the cost-to-go function is not piecewise linear, such approximations cannot be applied.

[2]Algorithm 2 uses two separate inputs, m and ℓ, to determine the number of sets constructed and the threshold on $W(\chi)$, respectively. This may seem redundant as $m = \ell$ here, but this distinction is necessary for our approximation algorithm (Algorithm 4) presented in Section V-B.

under any admissible control policy, the trace of the constructed MDP results in a trace of $Robot$ that satisfies ϕ. The root of the constructed MDP is (χ^0, b^0). The ith level of the tree is constructed from the $i-1$st level by enumerating for each state (χ, b) in the $i-1$st level the actions that can enable a transition from χ to a state $\chi' = (q', \sigma') \in$ automatonStates[i]. For each such action a, we enumerate all the possible measurements $y \in R_Y$ that can be observed in q' and calculate the estimate pmf b' that results from applying the Bayes filter (line 8) and add the state (χ', b') to the ith level. The probabilistic transition relation P_{tot} is then constructed (line 11).

Algorithm 3 Constructs the full model MDP

```
1: function ConstructMDP(P_φ,χ^0,b^0,automatonStates, ℓ)
2: MDPStates[0] := (χ^0, b^0)
3: for j := 1 to ℓ do
4:   for all (χ, b) ∈ MDPStates[j − 1],χ' ∈ automatonStates[j] do
5:     if χ ∉ F_{P_φ} then
6:       a := action such that (χ, a, χ') ∈ Δ_{P_φ}
7:       for all y ∈ R_Y do
8:         b' := output of (3) with b, a, y
9:         MDPStates[j].add((χ', b'))
10:        actionSet[(χ, b)][j − 1].add(a)
11:        P_tot((χ, b), a, (χ', b')) := output of (7)
12: return (MDPStates,P,actionSet)
```

Finally, the function BellmanIteration uses standard, finite-horizon Bellman iteration to calculate the optimal policy with zero stage costs.

The reactive policy generated by Algorithm 1 can be applied to $FullModel$ to generate trajectories of $Robot$ that are guaranteed to satisfy ϕ and in expectation minimize the Shannon entropy of the resulting estimate. The next theorem establishes the optimality of this method.

Theorem 1. *The policy generated by Algorithm 1 is the exact solution of Problem 1.*

Proof. See Appendix A □

Thus, Algorithm 1 minimizes the expected entropy under a budget of ℓ actions. We expect the performance of the algorithm to improve with a longer budget. Indeed,

Corollary 1. *Increasing the budget ℓ cannot decrease and can increase the expected quality of the calculated optimal policy.*

Proof. Appendix B. □

However, calculating longer policies is considerably more computationally expensive. More specifically,

Proposition 2. *The time complexity of Algorithm 1 is $O(|Act|^\ell |R_Y|^\ell |S|)$.*

The average case complexity can be lower than the bound given in Proposition 2 due to the pruning of actions that occurs when constructing $FullModel$ (Remark 2), but the exponential dependence on ℓ remains.

B. Receding Horizon Dynamic Programming

The computational cost of Algorithm 1 is too great for large budgets. As an alternative, Algorithm 4 constructs an approximation to the optimal policy on-line using a receding-horizon implementation. At each time step k, Algorithm 4 constructs an MDP whose root is the pair (χ^k, b^k) and whose depth is at most some finite horizon m (line 7). The ith level of each MDP is constructed from automaton state/pmf pairs that are reachable from (χ^k, b^k) in i transitions and can satisfy ϕ in $\ell - (k+i)$ actions. Bellman iteration is performed on this MDP to form the m-step optimal policy (Line 8). The agent applies the policy for $n \leq m$ time steps. Then, the agent begins the process again from an MDP with root (χ^{k+n}, b^{k+n}). This is repeated until the trajectory of $Robot$ has satisfied ϕ.

Algorithm 4 Receding-horizon approximation to Algorithm 1.

```
1: function RecedingHorizonDP(P_φ,χ^0,b^0 ,m,n,ℓ)
2: χ := χ^0; b := b^0;k := 0
3: if W(χ^0) > ℓ then
4:   return None
5: while χ ∉ F_{P_φ} do
6:   automatonStates := PossibleStates(P_φ,χ,m,ℓ − k)
7:   (MDPStates,P_tot,actionSet) := ConstructMDP(P_φ,χ,b,automatonStates)
8:   μ :=BellmanIteration(MDPStates,P_tot,actionSet)
9:   if k ≥ ℓ − m then
10:    n := ℓ − k
11:  for i := 1 to n do
12:    (χ, b) := result from applying μ(i, (χ, b))
13:    k + +
```

Proposition 3 establishes the time complexity of Algorithm 4, which leads to a significant savings in computational effort if ℓ is long.

Proposition 3. *The time complexity of Algorithm 4 with budget ℓ, planning horizon m, and action horizon n is $O(|Act|^m |R_Y|^m |S| \lceil \frac{\ell}{n} \rceil)$.*

Now, we must establish the correctness of Algorithm 4 with respect to the specification ϕ.

Theorem 2. *If $W(\chi^0) \leq \ell$,the solution of Algorithm 4 is guaranteed to satisfy ϕ.*

Proof. Appendix C □

Finally, we have to consider performance. Calculating a precise sub-optimality gap between Algorithms 1 and 4 is difficult due to the non-linear natures of the entropy measure and the Bayes filter and variability of possible automaton structures. Instead, we present a pair of results that establishes some general properties of the solution quality of Algorithm 4.

Proposition 4. *If $\ell = m$, executing Algorithm 4 and 1 result in the same policy.*

Corollary 2. *If $m < \ell$, the policies produced by Algorithm 4 are in general suboptimal.*

Proof. Appendix D. □

Intuition tells us that the performance should depend on the planning horizon m and the action horizon n, but similarly, we cannot characterize this relationship precisely. Instead, we will observe the effects of varying each of these quantities empirically in the next section.

C. Conclusion of running example

1) Comparison of Algorithms

We tested Algorithms 1 and 4 using a simulation of Example 1. We simulated 500 Monte Carlo trials of each method with horizon $\ell = 8$ on an 8-core ThinkPad with 2.1 GHz processors and 7.4 GB RAM. For the receding horizon method, the planning and acting horizons were $m = 3$, $n = 2$, respectively. The average terminal entropy $H(b^t)$ that resulted from the trajectories generated as well as the average error $e(t)$, defined as

$$e(t) = \sum_{j|q_j \in Q} | \sum_{\sigma|\sigma_j \neq s_j^t} b^t(\sigma) - s_j^t|, \tag{9}$$

for Algorithms 1 and 4 are given in Table I.

All trajectories generated by both algorithms satisfied the given scLTL specification. The resulting statistics for the simulation are given in Table I. The calculation time for Algorithm 1 was the total time required to calculate the optimal policy. For Algorithm 4, this is the average time required to calculate the optimal policy per trial. As expected, Algorithm 1 performed better (achieved lower average terminal entropy and error) than Algorithm 4. The difference in timing, however, was significant, as Algorithm 4 was faster by a factor of roughly 7.

Method	$H(b^t)$ (bits)	$e(t)$	Calculation time (s)
Algorithm 1	3.71	1.79	2.79
Algorithm 4	3.80	1.84	0.40 (per trial)

TABLE I
RESULTS FROM THE CASE STUDY SIMULATION USING 500 MONTE CARLO SIMULATIONS.

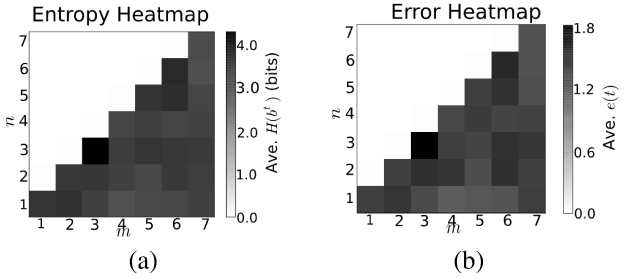


Fig. 4. Heatmaps of (a) the average terminal entropy $H(b^t)$ and (b) the average error scores $e(t)$ for 250 Monte Carlo trials at each pair of (m, n) values.

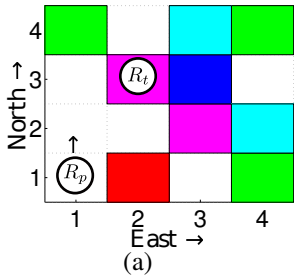
2) Effect of planning and action horizons

We also used this case study to evaluate empirically the effect of the planning horizon m and action horizon n on Algorithm 4. We varied the pairs (m, n) over the range $\{(m, n) \in \{1, \ldots, 7\}^2 | n \leq m\}$ and performed 250 Monte Carlo trials with each pair. The effect of varying m and n on average entropy and average error are shown in Figure 4(a) and (b),respectively.

The entropy and error both decrease overall as m grows larger and n grows smaller. These decreases, however, are not monotonic. We hypothesize that this non-monotonicity is not a sampling artifact. Rather, this relationship is likely heavily influenced by the transition probability of Env, the topology of the environment, and the specification ϕ.

VI. CASE STUDY

We complement the theoretical results and simulation of the running example from Section V with an experimental case study. The scenario is as shown below. We control a pursuing ground robot R_p that is tasked with localizing a target ground robot R_t moving according to a Markov chain (Figure 5a). That is, R_t can move to an adjacent cell if it is open with probability 0.15. The robots are controlled over a wireless network and their positions are tracked via the optiTrack motion capture system. R_p uses a (simulated) noisy camera to estimate the location of R_t (Figure 5a). Both robots are operating in a 4x4 labeled grid (Figure 5b).



(a)

Prop.	color
p_1	magenta
p_2	red
p_3	green
p_4	blue
p_5	cyan

(b)

Fig. 5. (a) Labeled, partitioned space in which the target robot (R_t) and pursuing robot (R_p) are operating. (b) Table showing the mapping between the color on the grid in (a) and the propositions used in specification (10).

A. Motion and measurement models

In addition to localizing R_t, R_p must move under the following constraints

$$\phi_{exp} = \quad \Diamond\, p_1 \wedge \Diamond\, p_3 \wedge (\neg p_3 U p_1) \wedge (\neg p_2 U p_3) \wedge ((\Diamond\, p_4) \Rightarrow (\Diamond\, p_5 \wedge (\neg p_3 \mathcal{U} p_5)). \tag{10}$$

The specification ϕ_{exp} can be translated to plain English as "Visit a magenta region (p_1) before visiting a green region (p_3). Always avoid the red region (p_2). If a blue region (p_4) is visited, then also visit a cyan region (p_5) before visiting a green region."

Figure 6 shows a small subset of the transition system $Robot$ constructed from the motion of R_p. As opposed to the scenario illustrated in Example 1, the orientation of R_p, in addition to its location, affects how well R_t can be sensed. Thus, the states of the transition system are given as triples (i, j, dir), where i and j are the East and North coordinates of the cell in which R_p is located and $dir \in \{N, S, E, W\}$ indicates the direction that R_p's camera faces. The propositions associated with each state do not depend on R_p's orientation. The actions R_p can take are $Act = \{CW, CCW, straight\}$, where $CW(CCW)$ is a clockwise (counter-clockwise) rotation of 90° and $straight$ moves R_p into the cell it is facing.

The noisy camera mounted on R_p returns measurements in the set $R_Y = \{None, Right, Center, Left\}$ where $None$ means R_t is not visible and $Right, Center, Left$ refer to which third of the field of vision of R_p is occupied by R_t. The measurement model is based on the relative positions of R_t and R_p and the orientation of R_p. The camera can detect R_t in a limited cone in front of it. The correct measurement rate falls off within the cone as the distance from the cone's center or R_p increases. The correct detection rate within a cell increases with the volume of the cone it contains. The correct detection rates are illustrated in Figure 7. These rates remain constant with respect to the relative orientations. The rate of false positives outside the cone is a constant 0.01 per each measurement $Right, Center, Left$.

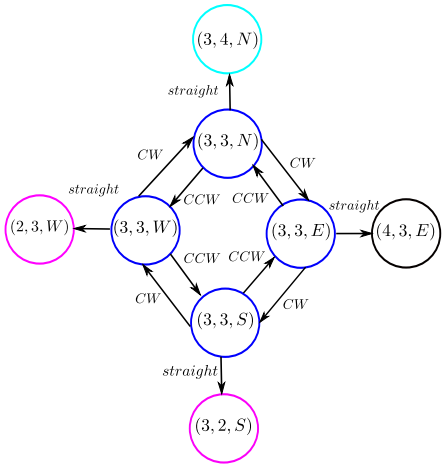


Fig. 6. A subsection of the transition system constructed from R_p.

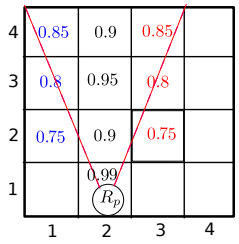


Fig. 7. Correct detection rates for the camera. Correct detection rates for $Center$ are shown in black, for $Left$ in blue, and for $Right$ in red.

B. Experimental validation

We used our experimental setup to evaluate 4 different policies:

1) a baseline "random walk" policy in which at each time i, an action is uniformly chosen from the set of actions that do not violate the constraints,
2) an optimal policy calculated from Algorithm 1 with budget $\ell = 8$,
3) a sub-optimal policy calculated from Algorithm 4 with budget $\ell = 8$, planning horizon $m = 4$, and action horizon $n = 1$, and
4) a longer policy calculated from Algorithm 4 with budget $\ell = 20$, planning horizon $m = 4$, and action horizon $n = 1$.

For each policy, we generated 250 Monte Carlo trials and implemented some select sample paths using the experimental testbed in our lab. The policies were calculated on a cluster with 32 2.1 GHz processors and 32GB RAM. The initial belief about the position of R_t was divided uniformly among the regions $(1,2),(2,3)$, and $(2,2)$. R_p was initialized in state $(1,1,N)$ and R_t was initialized in region $(2,3)$.

Paths that were generated by our system are shown in Figure 8. Only a few possible paths that satisfied (10) could be generated when $\ell = 8$. All paths are shown in Figure 8(a). Points in the trajectories where a policy can choose between actions are indicated with triangles. In total, there are 5 unique paths.

Many more possible paths can satisfy (10) when $\ell = 20$. We show part of an example sample path in Figure 8 (b)-(f) that illustrates how our methods balance information maximization with satisfying the temproal logic specification in time. Figure 8(b) shows the initial configuration of the agents. Note that R_t is initially within view of R_p. In Figure 8(c), R_p has moved forward and turned East to gather more information about the eastern side of the map that is not visible when facing North. Note that R_p is facing R_t at this point. R_p continues to track R_t eastward in Figure 8 (c)-(d). In Figure 8(c), R_t enters a region with label p_1, which means at this point it can satisfy (8) by avoiding p_2 (red) and going to p_3 (green). However, in Figure 8(d), R_p enters a region with label p_4, meaning it acquires the additional requirement to visit a region with label p_5 (cyan) before going to p_3. R_p chooses to satisfy the specification by visiting region $(4,2)$ rather than $(3,4)$ because it prefers to continue tracking R_t south, as shown in Figure 8(f). At this point, R_p has exhausted most of its budget, so it must choose whether to continue south or turn north to finish satisfying (10). R_p chooses to face East and then North as shown in Figure 8(g) in order to continue tracking R_t as it moves throughout the environment. Finally, R_p travels north to end its run in Figure 8(h).

Histograms of the terminal entropy $H(b^t)$ found using each policy are shown in Figure 9. Note that when compared to the random walk method, all of the other policies have a lower probability of ending with a high terminal entropy. It also appears that there is not an appreciable difference between the performance of Algorithms 1 and 4 when the budgets are the same. We did not investigate Algorithm 1 with $\ell > 8$ due to the state explosion problem. However, we can improve performance if we increase the budget for Algorithm 4.

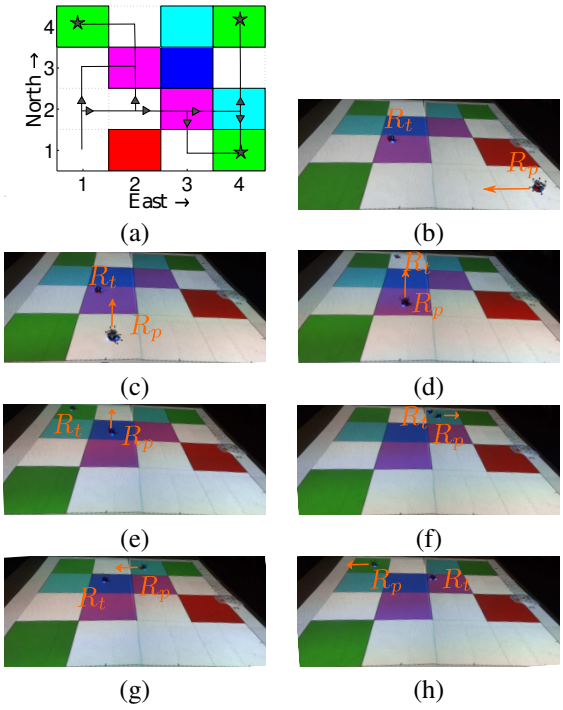


Fig. 8. (a) Schematic of all possible paths with $\ell = 8$. (b)-(h) Snapshots of sample path from applying Algorithm 4 with $\ell = 20$.

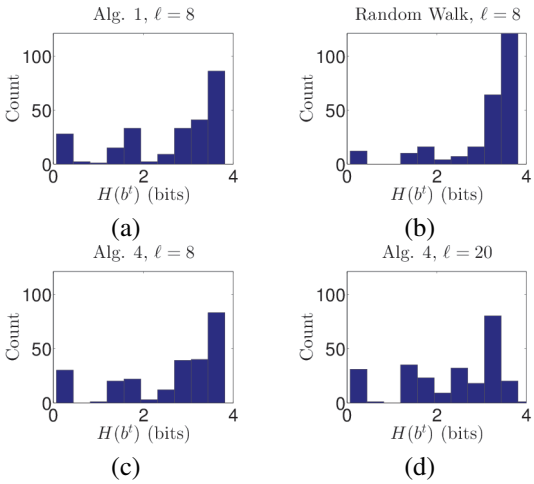


Fig. 9. Histograms of terminal entropy $H(b^t)$ resulting from 250 Monte Carlo trials using (a) dynamic programming (DP) (b) a random walk (c) receding horizon DP with a short budget (d) receding horizon DP with a long budget.

Method	ℓ	$H(b^t)$ (bits)	Calculation time
R. Walk	8	3.0660	$<$ 1s
Alg. 1	8	2.6323	3627s
Alg. 4	8	2.6371	34s/trial
Alg. 4	20	2.3557	543s/trial

TABLE II
RESULTS FROM EXPERIMENTS USING 4 DIFFERENT METHODS.

The results in Table II confirm our observations about Figure 9. All of our methods significantly outperform the random walk method. Note, too, that the difference in performance between Algorithm 1 and Algorithm 4 when $\ell = 8$ is negligible, but Algorithm 4 is faster by a factor of greater than 10. This means that, at least in some situations, Algorithm 4 approximates Algorithm 1 very closely. Further, when we increased the budget from $\ell = 8$ to $\ell = 20$, the computation time of Algorithm 1 slowed down by a factor of 16, but the performance increased appreciably. Meanwhile, Algorithm 1 had significant difficulties with the increase from $\ell = 8$ to $\ell = 9$. These results indicate that our receding horizon approximation is promising for use in long-term, persistent monitoring applications.

VII. CONCLUSIONS

In this paper, we have formulated a path planning problem which considers both the expected informativeness of measurements along a given path and the satisfaction of high-level specifications of temporal logic formulae. We showed how to construct a Markov decision process that models the simultaneous evolution of the robot's motion, its progress towards satisfaction of a given temporal logic formula, and its estimate of an unknown feature. We connected the constrained informative path-planning problem to stochastic optimal control and used two stochastic dynamic programming methods to solve the problem: a provably optimal off-line approach and a receding-horizon approximation that can be calculated with lower computational complexity. Our results from simulation and experimentation indicate that the receding horizon method reduces entropy reasonably well when compared to the off-line, optimal dynamic programming method.

We provide experimental evidence that our methods are able to be applied to physical robotics systems. We plan to extend our implementation to robots operating in less structured indoor environments using real sensor data on-line.

We plan to extend our receding horizon approximation to consider problems of persistent monitoring. Our preliminary experimental results indicate that the required computation time scales well as the budget increases. Other future directions in this area include extending our methods to multi-agent path planning and also considering more carefully the continuous dynamics and measurements of the agents.

APPENDIX

A. Proof of Theorem 1

Proof. Let $\mu^*_{\chi^0,b^0,\ell,\ell}$ (μ^*, briefly) be the result of applying Algorithm 1 to $MDP_{\chi^0,b^0,\ell,\ell}$ (MDP_ℓ, briefly), the MDP constructed in Algorithm 3. μ^* is calculated by applying finite horizon Bellman iteration to MDP_ℓ, so, by principle of optimality,

$$\mu^* = \arg\min_{\mu \in M_\ell} E_{(MDP_\ell,\mu)}[H(b^t)], \tag{11}$$

where $M_{\chi^0,b^0,\ell,\ell}$ (M_ℓ) is the set of all possible policies of length ℓ that can be defined over MDP_ℓ. Thus in order to prove the optimality of μ^*, we must prove that the set of traces produced by MDP_ℓ under the policies in M_ℓ are exactly those traces which satisfy the constraints (8b)-(8e).

(8b) The max depth of MDP_ℓ is ℓ.

(8c) In order for an action a to be in actionSet$[(\chi, b)][i]$, (a) $(\chi, a, \chi') \in \Delta_{\mathcal{P}_\phi}$ and (b) $P_{est}(b, q', b') > 0$. (a) and (b) together imply $P_{tot}((\chi, b), a, (\chi', b')) > 0$, meaning (8c) is satisfied for all $a \in$ actionSet$[\cdot][i]$, $i = 0 \dots \ell - 1$.

(8d) This follows immediately from (a) and (b) and Algorithm 3, line 11.

(8e) Since $\chi \in$ automatonStates$[i]$ $\forall(\chi, b) \in$MDPStates$[i]$, every state in MDPStates$[i]$ is at most $\ell - i$ transitions away from a state $(\chi_f, b_f) \in F_{\mathcal{P}_{phi}} \times B$. Every state $\chi \in$ automatonStates$[i]$ with distance to acceptance $W(\chi) < \ell - i$ has a neighbor that is closer to an accepting state, so $\exists \chi' \in$ automatonStates$[i+1]$ such that $W(\chi') < \ell - i - 1$. Every state in MDPStates$[i]$ is either an accepting state or has a child in MDPStates$[i+1]$. Since MDPStates$[\ell]$ is constructed from automatonStates$[\ell]$, either each state in MDPStates$[\ell]$ is accepting or MDPStates$[\ell] = \emptyset$, implying $\exists k < \ell$ such that every state in MDPStates$[k]$ is accepting. Therefore, every trace produced by MDP_ℓ must end in accepting state.

□

B. Proof of Corollary 1

Proof. Let $\ell_1 < \ell_2$ and let $\mu_i = \mu^*_{\chi^0,b^0,\ell_i,\ell_i}$ Since the states and actions in MDP_{ℓ_1} at each level are included

in the states and actions in MDP_{ℓ_2} at the same level for the first ℓ_1 levels, $M_{\ell_1} \subseteq M_{\ell_2}$. Thus,

$$\min_{\mu \in M_{\ell_1}} E_{(MDP_{\ell_1},\mu)}[H(b^t)] \geq \min_{\mu \in M_{\ell_2}} E_{(MDP_{\ell_1},\mu)}[H(b^t)]. \tag{12}$$

Further, since the family of possible traces produced by MDP_{ℓ_1} under μ_2 is a subset of the family of possible traces produced by MDP_{ℓ_2} under μ_2,

$$E_{(MDP_{\ell_1},\mu_2)}[H(b^t)] \geq E_{(MDP_{\ell_2},\mu_2)}[H(b^t)]. \tag{13}$$

Inequalities (12) and (13) together imply the corollary. □

C. Proof of Theorem 2

Proof. At time $k < \ell - m$, if the full state of the system is described by (χ^k, b^k), Algorithm 4 constructs $MDP_{\chi^k,b^k,\ell,m}$ (line 7). If we apply any policy $\mu \in M_{\chi^k,b^k,\ell,m}$ for n time periods, we are guaranteed to visit a sequence of states (χ^{k+i}, b^{k+i}) such that $W(\chi^{k+i}) \leq \ell - k - i \;\; i = 1, \ldots, n$. Thus while $k < \ell - m$, $W(\chi^k) \leq \ell - k$. If Algorithm 4 has not terminated when $k \geq \ell - m$, Algorithm 4 constructs $MDP_{\chi^k,b^k,\ell-k,\ell-k}$, and applies the policy μ^*, effectively executing Algorithm 1 with initial state (χ^k, b^k) and horizon $\ell - k$. Since applying Algorithm 4 up until time k guarantees $W(\chi^k) < \ell - k$, by Theorem 1, *Robot* will satisfy ϕ within $\ell - k$ steps. □

D. Proof of Corollary 2

Proof. Let $\sharp$ be the policy concatenation operator such that for two policies μ_1, μ_2, $\mu_1[n_1]\sharp\mu_2[n_2]$ is the policy resulting from applying μ_1 for n_1 time steps and μ_2 for n_2 time steps. Also define $[\cdot], \sharp$ set wise such that if M_1, M_2 are sets of policies, then $M_1[n_1]\sharp M_2[n_2] = \{\mu_1[n_1]\sharp\mu_2[n_2] | \mu_1 \in M_1, \mu_2 \in M_2\}$. Let $M^{RH}_{\ell,m,n}$ be the family of policies considered in Algorithm 4. Then,

$$\begin{aligned} M^{RH}_{\ell,m,n} = \; & \{M_{\chi^0,b^0,\ell,m}[n]\sharp M_{\chi^n,b^n,\ell,m}[n]\sharp \\ & \ldots\sharp M_{\chi^{pn},b^{pn},\ell-pn,\ell-pn} \\ & |(\chi^{rn}, b^{rn}) \in MDP_{\chi^{(r-1)n},b^{(r-1)n},\ell,m} \\ & \forall r = 1, \ldots, p\} \end{aligned} \tag{14}$$

where $\ell - pn \leq m$. Thus, $\mu^* \in M^{RH}_{\chi^0,b^0,\ell,m,n}$, but $\mu^{RH}_{\chi^0,b^0,\ell,m,n} \neq \mu^*$ in general. Since $M^{RH}_{\chi^0,b^0,\ell,m,n} \subseteq M_\ell$, no other policy in $M^{RH}_{\chi^0,b^0,\ell,m,n}$ can outperform it. Therefore Algorithm 4 with $m < \ell$ is suboptimal.

□

REFERENCES

[1] I.F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102 – 114, aug 2002.

[2] 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.

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

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

[5] 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.

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

[7] J. Binney, A. Krause, and G.S. Sukhatme. Informative path planning for an autonomous underwater vehicle. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4791 –4796, May 2010.

[8] S. Candido and S. Hutchinson. Minimum uncertainty robot navigation using information-guided pomdp planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 6102 –6108, may 2011.

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

[10] Xu Chu Ding, Mircea Lazar, and Calin Belta. Receding horizon temporal logic control for finite deterministic systems. In *American Control Conference (ACC)*, Montreal, Canada, 2012.

[11] Seng Keat Gan, R. Fitch, and S. Sukkarieh. Real-time decentralized search with inter-agent collision avoidance. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 504 –510, May 2012.

[12] A. Ghaffarkhah and Y. Mostofi. Path planning for networked robotic surveillance. *Signal Processing, IEEE Transactions on*, 60(7):3560 –3575, july 2012.

[13] 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.

[14] L. P. Kaelbling, M.L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence in Medicine*, 101:99–134, 1998.

[15] S. Karaman and E. Frazzoli. In *American Control Conference (ACC), 2012*, pages 735–742.

[16] A. Kassir, R. Fitch, and S. Sukkarieh. Decentralised information gathering with communication costs. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2427 – 2432, may 2012.

[17] 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.

[18] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *IEEE International Conference on Robotics and Automation*, pages 3116–3121, 2007.

[19] Hadas Kress-Gazit, Gerogios E. Fainekos, and George J. Pappas. Temporal logic based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

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

[21] 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.

[22] Morteza Lahijanian, Sean B. Andersson, and Calin Belta. Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Transaction on Robotics*, 28:396–409, 2011.

[23] Timo Latvala. Efficient model checking of safety properties. In Thomas Ball and Sriram Rajamani, editors, *Model Checking Software*, volume 2648 of *Lecture Notes in Computer Science*, pages 624–636. Springer Berlin / Heidelberg, 2003.

[24] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36, pages 789-814, 1999.

[25] Alexandra Meliou, Andreas Krause, Carlos Guestrin, and Joseph M. Hellerstein. Nonmyopic informative path planning in spatio-temporal models. In *Proceedings of the 22nd national conference on Artificial intelligence*, volume 1, pages 602–607, 2007.

[26] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. IJCAI.

[27] S. Joe Qin and Thomas A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11, pages 733-764, 2003.

[28] Riccardo and Scattolini. Architectures for distributed and hierarchical model predictive control a review. *Journal of Process Control*, 19(5):723 – 731, 2009.

[29] M. Schwager, P. Dames, D. Rus, and V. Kumar. A multi-robot control policy for information gathering in the presence of unknown hazards. In *Proceedings of the International Symposium on Robotics Research (ISRR 11)*, Aug. 2011.

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

[31] Amarjeet Singh, Andreas Krause, Carlos Guestrin, William Kaiser, and Maxim Batalin. Efficient planning of informative paths for multiple robots. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2204–2211, 2007.

[32] A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In C. J. Tomlin and M. R. Greenstreet, editors, *Hybrid Systems: Computation and Control HSCC*, volume 2289 of *LNCS*, pages 465–478. Springer, March 2002.

[33] A. Ulusoy, T. Wongpiromsarn, and C. Belta. Incremental control synthesis in probabilistic environments with temporal logic constraints. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 7658–7663, 2012.

[34] 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.

[35] Cristian Vasile and Calin Belta. Sampling-Based Temporal Logic Path Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[36] Tichakorn Wongpiromsarn and Emilio Frazzoli. Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications. *CoRR*, abs/1203.1177, 2012.

[37] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard Murry. Receding horizon control for temporal logic specifications. In *HSCC10: Proceedings of the 13th ACM International Conference on Hybrid Systems:Computation and Control*, pages 101–110, 2010.

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