2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Temporal Logic Motion Planning in Unknown Environments

A. I. Medina Ayala, S. B. Andersson, and C. Belta

Abstract— **In this paper, we consider a robot motion planning problem from a specification given as a syntactically co-safe linear temporal logic formula over a set of properties known to be satisfied at the regions of an unknown environment. The robot is assumed to be equipped with deterministic motion and accurate sensing capabilities. The environment is assumed to be partitioned into a finite number of identical square cells. By bringing together tools from formal verification, graph theory, and grid-based exploration, we develop an incremental algorithm that makes progress towards satisfying the specification while the robot discovers the environment using its local sensors. We show that the algorithm is sound and complete. We illustrate the feasibility and effectiveness of our approach through a simulated case study.**

I. INTRODUCTION

The flexibility of model checking and automata game techniques, and the proven expressivity of temporal logics allow the use of formal methods to extend the applicability of classical robotics problems. In particular, one of the areas within the robotics community that has greatly benefited from the use of formal methods is path planning. Temporal logics such as the Computation Tree Logic (CTL) [1], Continuous Stochastic Logic (CSL) [2], Linear Temporal Logic (LTL) [3], and μ-calculus [4], have been effectively applied to express complex high-level planning specifications. Furthermore, the adaptation of existing off-the-shelf model checking and automaton based tools makes it possible to automatically generate the solution to the path planning problem from such specifications.

In general, solving the path planning problem by means of formal methods requires *a priori* knowledge of the robot's workspace. There are many applications, such as in search and rescue operations, where the robot must deal with an unknown or partially known environment. Despite this lack of knowledge, the robot may be required to plan a strategy that fulfills certain requirements based on the mission at hand. Consider, for instance, a robot deployed in a building after an earthquake. Prior information about the building may be available in the form of, for example, a blueprint, but the disaster is likely to have significantly altered the environment. The task given to the robot could be to look for survivors, guide them while avoiding unsafe areas, and eventually bring them to safe areas. Given that the environment is only partially known, the robot will need to gather information about it in real time and use that while planning so as to satisfy the task specification.

The example described above is an instance of the Simultaneous Localization and Mapping (SLAM) [5] problem. Motivated by a wide number of applications that fall into the SLAM framework, this paper proposes an algorithm that interleaves the use of formal methods, graph theory and classical exploration techniques to solve the path planning problem in an unknown environment subject to temporal logic specifications. Specifically, given a known set of labels capturing certain elements of interest that can be discovered during the robot's navigation through an unknown environment and a temporal logic formula over this set of labels, we obtain a trajectory that satisfies the formula.

Our approach to solving the planning problem takes advantage of automata-based model checking [6] and runtime verification techniques [7]. We exploit the automata-based model checking framework to find a trajectory satisfying the formula given the current information about the environment. However, the incomplete knowledge of the environment potentially has a negative effect on finding such a trajectory. Even though the formula is not yet satisfied, there may exist at least one trajectory that does not violate the formula. We use the runtime verification setting to monitor a set of potential trajectories that lead to unexplored areas of the environment and select the path that does not violate the formula, provided one exists, and is the most promising in terms of the satisfaction of the formula.

Our work is related to the problem of reactive synthesis, in which a finite state machine satisfying a desired output behavior subject to a temporal logic constraint is generated regardless of the input applied [8]. In particular, [9] presents an approach to automatically synthesize a hybrid controller that guarantees a user-defined specification while exploring a partially known environment. As new regions of the environment are detected, the specification is rewritten and re-synthesized. Related work includes also [10], which considers the synthesis of controllers in environments with uncertain, but fixed structure. By locally modifying a nominal plan if it fails, the controller is able to deal with unexpected changes in the environment. These works are restricted to the class of Generalized Reactivity (GR)(1) formulas [11]. Unlike these approaches, in this work we use syntactically co-safe LTL formulas to express the mission specifications. Syntactically co-safe LTL formulas not only describe finite horizon specifications, expressing a wide spectrum of high-level robotic missions, but also belong to

This work is partially supported at Boston University by the NSF under grants CNS-1035588 and CMMI-0928776, and the ONR MURI under grant N00014-09-1051.

Medina Ayala is with the Department of Mechanical Engineering, Andersson and Belta are with the Department of Mechanical Engineering and the Division of Systems Engineering, Boston University, MA, USA, E-mail: duvinci@bu.edu, sanderss@bu.edu, cbelta@bu.edu

A. Medina Ayala is the corresponding author.

the class of languages that are monitorable [12].

Other related work includes [13] where the control synthesis problem on a graph is constrained to maximize the accumulated reward locally while satisfying an LTL mission specification. In order to solve this problem, a receding horizon controller is devised to guarantee the fulfillment of the specification in infinite time. Even though this work can locally synthesize a control strategy while satisfying the specification, it still requires knowledge of the graph representing the workspace *a priori*. In contrast, our approach incrementally builds the transition system describing the motion of the robot while exploring the environment to find a path satisfying a given specification.

II. PRELIMINARIES

Definition 1. *A weighted finite deterministic transition system (TS) is a tuple $\mathcal{T} = (S, s_0, \Delta, w, \Pi, l)$ where S is a finite set of states, $s_0 \in S$ is the initial state, $\Delta \subseteq S \times S$ is the set of transitions, $w : \Delta \to \mathbb{N}$ is a weight function that assigns a positive value to each transition, Π is a set of observations, and $l : S \to 2^{\Pi}$ is the labeling map.*

For convenience of notation, we use $s \to_{\mathcal{T}} s'$ if $(s, s') \in \Delta$. A finite trajectory of a TS is a finite sequence $\tau = s_0 s_1 \dots s_n$, where $s_k \to_{\mathcal{T}} s_{k+1}$ for all $0 \leq k \leq n-1$. The finite trajectory τ generates a finite word $\pi = \pi_0 \pi_1 \dots \pi_n$, where $\pi_k = l(s_k)$ for all $k = 0, \dots n$.

Definition 2. *[14]. A syntactically co-safe LTL (scLTL) formula over a set of atomic propositions Σ is inductively defined as follows:*

$$\Phi ::= \sigma \mid \neg\sigma \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \mathcal{X}\ \Phi \mid \Phi\ \mathcal{U}\ \Phi \mid \mathcal{F}\ \Phi$$

where $\sigma \in \Sigma$ is an atomic proposition, $\neg$ (negation), $\vee$ (disjunction), and $\wedge$ (conjunction) are Boolean operators, and $\mathcal{X}$ (next), $\mathcal{U}$ (until), and $\mathcal{F}$ (eventually) are temporal operators.

The semantics of scLTL formulas are defined over infinite words in 2^{Σ}. Intuitively, $\mathcal{X}\ \sigma$ asserts that σ becomes true in the next position in the word; $\sigma_1\ \mathcal{U}\ \sigma_2$ expresses that σ_1 is true until σ_2 becomes true in a word; and $\mathcal{F}\ \sigma$ states that σ becomes true at some position in the word. More complex specifications can be expressed by combining Boolean and temporal operators (see, e.g. (3)).

Let $\mathcal{L}(\phi)$ be the set of infinite words satisfying an scLTL formula ϕ. $\mathcal{L}(\phi)$ can be described by the deterministic finite automaton defined as follows.

Definition 3. *A deterministic finite automaton (DFA) is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is the input alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states.*

We use $q \xrightarrow{\sigma}_{\mathcal{A}} q'$ if $q' = \delta(q, \sigma)$. An accepting run $\tau_{\mathcal{A}}$ of an automaton $\mathcal{A}$ on a finite word $\sigma_0 \sigma_1 \dots \sigma_d$ over Σ is a sequence of states $\tau_{\mathcal{A}} = q_0 q_1 \dots q_{d+1}$ such that $q_0 \in Q$, $q_{d+1} \in F$, and $\delta(q_k, \sigma_k) = q_{k+1}$ for all $k = 0, \dots d$.

Model checking on the TS $\mathcal{T}$ for an scLTL formula ϕ can be conducted by the parallel composition between $\mathcal{T}$ and a DFA $\mathcal{A}$ that accepts all runs satisfying ϕ.

Definition 4. *Given a TS $\mathcal{T} = (S, s_0, \Delta, w, \Pi, l)$ and a DFA $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$, their weighted product automaton is a DFA $\mathcal{A}^{\mathcal{P}} = (Q_P, q_{P0}, \delta_P, w_P, F_P)$, where $Q_P = S \times Q$ is the set of states, $q_{P0} = (s_0, q_0)$ is the initial state, $\delta_P \subseteq Q_P \times Q_P$ is the set of transitions defined by $((s, q), (s', q')) \in \delta_P$ if and only if $s \to_{\mathcal{T}} s'$ and $q \xrightarrow{l(s)}_{\mathcal{A}} q'$, $w_P : Q_P \times Q_P \to \mathbb{N}$ is a weight function such that $w_P((s_i, q_k), (s_j, q_l)) = w(s_i, s_j)$, where $(s_j, q_l) \in \delta_P((s_i, q_k))$, and $F_P = S \times F$ is the set of final states.*

An accepting run $\tau_P = (s_0, q_0) \dots (s_n, q_n)$ of $\mathcal{A}^{\mathcal{P}}$ defines an accepting run $q_0 \dots q_n$ of $\mathcal{A}$ over the input word $l(s_0) \dots l(s_{n-1})$.

For a DFA $\mathcal{A}$, let $\mathcal{A}(q)$ be an identical DFA except for the initial state, which is redefined in $\mathcal{A}(q)$ as $q_0 = q$. Let ϕ be an scLTL formula over 2^{Σ}, and let $\mathcal{A}^{\phi} = (Q^{\phi}, q_0^{\phi}, \Sigma, \delta^{\phi}, F^{\phi})$ be the DFA accepting all the words satisfying ϕ; i.e., $\mathcal{L}(\phi)$. Also, let $\mathcal{A}^{\neg\phi} = (Q^{\neg\phi}, q_0^{\neg\phi}, \Sigma, \delta^{\neg\phi}, F^{\neg\phi})$ be the DFA accepting all the words falsifying ϕ; i.e., $\mathcal{L}(\neg\phi)$. Let $u \in \Sigma^*$ be a finite word. u is a good prefix for ϕ if $\forall \sigma \in \Sigma^{\omega}$, $u\sigma \in \mathcal{L}(\phi)$. On the other hand, u is a bad prefix for ϕ if $\forall \sigma \in \Sigma^{\omega}$, $u\sigma \in \mathcal{L}(\neg\phi)$. Furthermore, u is an inconclusive prefix for ϕ if and only if for all $\forall \sigma \in \Sigma^{\omega}$, $u\sigma$ is neither a good nor a bad prefix.

A monitor is obtained by defining a Finite State Machine (FSM) constructed as follows. For the automaton $\mathcal{A}^{\phi}$, the function $\mathcal{F}^{\phi} : Q^{\phi} \to \mathbb{B}$ (with $\mathbb{B} = \{\top, \bot\}$) is defined. The set $\mathcal{F}^{\phi}(q) = \top$ if and only if $\mathcal{L}(\mathcal{A}^{\phi}(q)) \neq \emptyset$; a state q evaluates to $\top$ if and only if the language of the automaton starting in state q is not empty. Using $\mathcal{F}^{\phi}$, let $\hat{\mathcal{A}}^{\phi} = (Q^{\phi}, q_0^{\phi}, \Sigma, \delta^{\phi}, \hat{F}^{\phi})$ be the DFA with $\hat{F}^{\phi} = \{q \in Q^{\phi} | \mathcal{F}^{\phi}(q) = \top\}$. Analogously, set $\hat{\mathcal{A}}^{\neg\phi} = (Q^{\neg\phi}, q_0^{\neg\phi}, \Sigma, \delta^{\neg\phi}, \hat{F}^{\neg\phi})$ with $\hat{F}^{\neg\phi} = \{q \in Q^{\phi} | \mathcal{F}^{\neg\phi}(q) = \top\}$. Then,

Definition 5. *Given $\hat{\mathcal{A}}^{\phi} = (Q^{\phi}, q_0^{\phi}, \Sigma, \delta^{\phi}, \hat{F}^{\phi})$ and $\hat{\mathcal{A}}^{\neg\phi} = (Q^{\neg\phi}, q_0^{\neg\phi}, \Sigma, \delta^{\neg\phi}, \hat{F}^{\neg\phi})$, we define the FSM $\mathcal{M} = (\bar{Q}, \bar{q}_0, \bar{\delta}, \bar{\lambda})$, where $\bar{Q} = Q^{\phi} \times Q^{\neg\phi}$ is a finite set of states, $\bar{q}_0 = (q_0^{\phi}, q_0^{\neg\phi})$ is the initial state, $\bar{\delta}((q, q'), a) = (\delta^{\phi}(q, a), \delta^{\neg\phi}(q', a))$ is the transition function, and $\bar{\lambda} : \bar{Q} \to \mathbb{B}_3$ is the labeling function defined by*

$$\bar{\lambda}(q, q') = \begin{cases} \top & \text{if } q' \notin \hat{F}^{\neg\phi} \\ \bot & \text{if } q \notin \hat{F}^{\phi} \\ ? & \text{if } q \in \hat{F}^{\phi} \text{ and } q' \in \hat{F}^{\neg\phi}. \end{cases}$$

The defined FSM $\mathcal{M}$ is the monitor of an scLTL formula that yields $\top$, $\bot$ or ? for a word that is a good, bad, or inconclusive prefix, respectively.

III. PROBLEM FORMULATION AND APPROACH

In this work, we consider a ground mobile robot deployed in an unknown planar environment containing a known set of labels. These labels capture the presence of certain elements of interest within the environment. For simplicity of presentation and to keep the discussion focused on the

algorithmic part of the problem, the underlying framework used in this work is a partitioning of the environment into finitely many identical square cells storing the occupancy condition [15] (either obstacle free or occupied) and the labels holding true in each cell.

Given the discrete nature of the environment model, the robot's motion is also discretized. More specifically, the robot can move from its current cell to any of the adjacent neighboring cells provided they are not occupied by an obstacle. In principle, this assumption can be applied to robots with realistic dynamics, such as unicycles, cars, and quadrotors by solving input-output linearization problems, partitioning the robot's workspace using simplicial and rectangular partitions, and assigning vector fields in the regions of the partition [16], [17].

Initially the robot knows nothing about the environment except its current cell and the set of labels it may encounter within the environment. Assuming the robot has perfect localization, it can determine its current cell exactly. Then, the robot gains information about the environment by means of its sensors. Every time a new sensor reading is obtained, all the cells within the sensors' range are updated and integrated into the robot's knowledge. Such an update includes the accurate identification of the labels corresponding to each cell within the sensor's range, as well as their occupancy condition. The robot maintains the currently known map and the history of each one of the cells it has moved through starting from the initial cell. Hence, we consider the following problem.

Problem 1. *Given a mobile robot deployed in an unknown partitioned environment with a known set of labels, plan a trajectory that satisfies a syntactically co-safe linear temporal logic specification over this set, if one exists, or determine that one does not.* □

Our approach to *Problem 1* relies on modeling the motion of the robot within the environment as a TS $\mathcal{T}$, (see Def. 1). Each state of $\mathcal{T}$ corresponds to a cell in the partition with an assigned set of labels. Notice that $\mathcal{T}$ captures only the segment of the environment that has been explored so far. A robot trajectory is defined as the finite sequence of states of $\mathcal{T}$ that have been traversed since the robot's deployment in the environment.

The main idea behind the solution to *Problem 1* builds on the combination of automata-based model checking, monitor-based runtime verification, and frontier-based exploration methods. We incrementally construct a weighted product automaton $\mathcal{A}^{\mathcal{P}}$ (see Def. 4) by adding the new information the robot obtains from the environment. After each update, an optimal accepting run of the product automaton is sought by means of the graph algorithm presented in [18]. If no solution is reported based on the the partially known map, frontier cells are determined. A frontier cell is an obstacle-free cell that is adjacent to at least one "unknown" cell [19] (see Fig. 1). Adjacent frontier cells are grouped into frontier regions. Then, the robot calculates the shortest traveling distance to a representative cell of each frontier region by means of a search algorithm [20]. Once the shortest obstacle-free paths to each frontier region are obtained, the word generated by each one of them is analyzed by a monitor of the scLTL formula representing the robot's specification. As a result, the minimum number of letters to be appended to each word in order to satisfy the specification is obtained. The robot then moves to the frontier region that minimizes a cost function combining the traveling distance and the monitor function's output, by following the path given by the search algorithm. After achieving the selected frontier cell, the robot measures its environment and updates the map of the environment and the product automaton by inserting new edges and nodes in the automaton graph. The cycle is repeated until either a satisfying trajectory is found or the environment has been completely explored, i.e., no more frontier regions can be detected.

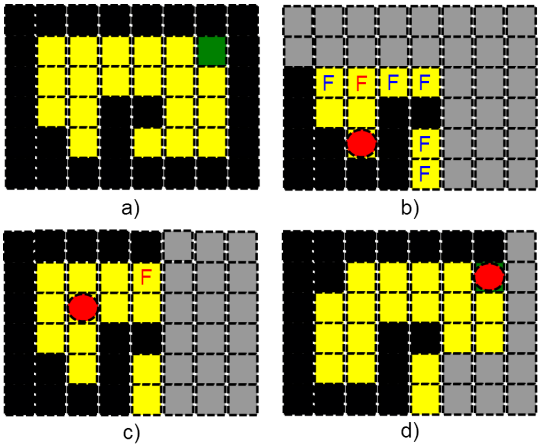

Fig. 1. A schematic representation of a robot's grid-world, which consists of *Free* cells (in yellow), *Obstacles* (in black), and a *Goal* (in green). The robot is represented as a red disk. Fig. a) depicts the actual topological structure of the environment. Figs. b) - d) represent the knowledge gained by the robot about the environment while accomplishing its task specification. The gray cells correspond to unknown cells and the cells labeled with "F" represent frontier cells.

IV. MOTION PLANNING FRAMEWORK

As outlined in Sec. III above, our approach attempts to find a trajectory that satisfies a temporal logic specification expressed as an scLTL formula ϕ in an *a priori* unknown environment. As seen in Alg. 1, we divide our approach into the initialization and the trajectory search procedures.

Lines 1-4 of Alg. 1 are run only once and are responsible for the *Initialization* of the algorithm. In line 2 of Alg. 1, we construct the DFA $\mathcal{A}$ that captures the temporal logic specification ϕ as in Def. 3. In line 3 of Alg. 1, we construct the FSM $\mathcal{M}$ representing the monitor of the specification as defined in Def. 5.

The rest of the algorithm (lines 5-14) is executed until either a trajectory satisfying ϕ is found, there are no more frontier regions left in the environment, or all the paths to the frontier regions violate the specification. The latter case implies that the robot is not able to satisfy the specification. The procedure *Find Path* takes as inputs the DFA $\mathcal{A}$, the

FSM $\mathcal{M}$ obtained in the initialization procedure, and the set $\mathcal{C}$ of new cells detected by the robot's sensor and gives as an output a trajectory to be executed by the robot. This can be achieved by dividing the procedure into three stages: transition system update, incremental automaton-based search, and frontier-based exploration.

The transition system update (line 7 of Alg. 1) incorporates the newly discovered information about the environment, represented by $\mathcal{C}$ to the model representing the motion of the robot. The updated transition system is then used to construct the product automaton incrementally in line 8 of Alg. 1 and an optimal robot trajectory is sought. If this search was unsuccessful (lines 10-12 of Alg. 1), a trajectory is planned by means of the frontier-based exploration stage. In what follows, each one of these stages is fully described.

Algorithm 1 The temporal logic path planning algorithm

1: **procedure** INITIALIZATION(ϕ)
2: Construct DFA $\mathcal{A}$ corresponding to ϕ
3: Construct FSM $\mathcal{M}$ corresponding to ϕ
4: **end procedure**

5: **procedure** FIND PATH($\mathcal{A}, \mathcal{M}, \mathcal{C}$)
6: **repeat**
7: $\mathcal{T} \leftarrow$ **Transition System Update**($\mathcal{C}$)
8: $\mathcal{A}^{\mathcal{P}} \leftarrow$**Incremental Automaton-Based Search**($\mathcal{T}$, $\mathcal{A}$)
9: **return** τ
10: **if** $\tau \not\models \phi$ **then**
11: **Frontier-Based Exploration**($\mathcal{C}$, $\mathcal{M}$)
12: **end if**
13: **until** termination conditions are satisfied
14: **end procedure**

A. Transition System Update

Under our framework, given the absence of *a priori* knowledge of the robot's environment, the complete transition system is not accessible. The construction of the transition system $\mathcal{T}$ is performed in an incremental fashion. $\mathcal{T}$ models the motion of the robot within the cells of the environment that have been identified so far. We initialize the set of states of $\mathcal{T}$ as a single state corresponding to the robot's current cell. An update in $\mathcal{T}$ is the outcome of the robot's local sensing providing a new set of states to be included in $\mathcal{T}$ corresponding to the set of cells within the robot's sensor range. Moreover, we define unit weight transitions between all adjacent cells in this range and identify their corresponding labels. Thus, the updated transition system inherits the set of transitions, weights, and labels from the recently discovered cells.

As an example, consider the environment illustrated in Fig. 1.a). A robot deployed without having any prior information in such an environment starts by measuring the cells within its sensors' range (Fig. 1.b)). The robot's current knowledge about the environment corresponds to all non-gray colored cells. Once the robot travels to a new cell such as in Fig. 1.c), its transition system includes the currently and previously seen cells.

B. Incremental Automaton-Based Search

Naively, searching for a trajectory satisfying an scLTL formula ϕ when only partial information about the system is unveiled can be achieved by constructing a new product automaton each time $\mathcal{T}$ is updated. In order to alleviate the time complexity of this process, and to take advantage of recent advances in graph algorithms, we integrate a method based on the Incremental Breadth First Search (IBFS) algorithm [18] into our framework.

The algorithm starts by defining two types of trees, trees rooted from the automaton's initial state q_{P0}, and trees rooted into every vertex $q \in F_P$. After the transition system has been updated, the algorithm scans the tree rooted from the automaton's initial state q_{P0} and identifies the path $q_0 - q_{cur}$, where q_{cur} is the state of the automaton corresponding to the current's robot cell. Then, this tree is expanded by scanning the set of new edges generated after the transition system update and augmenting the arcs rooted from q_{cur}. If no arc is found, the algorithm terminates, otherwise the expansion continues from the newly added set of vertices. Similarly, the trees rooted into vertices $q \in F_P$ are also augmented and expanded if possible. If an arc (q, q') with q rooted from $q_{cur} - q$ and q' rooted into a vertex in F_P is discovered, the path obtained by concatenating the $q_{cur} - q$ path, the arc (q, q'), and the $q' - q''$ path, with $q'' \in F_P$ is an optimal trajectory satisfying ϕ. It can be shown that such a path is also the shortest path [18].

C. Frontier-Based Exploration

The frontier-based exploration stage is triggered by the absence of an accepting run in the product automaton $\mathcal{A}^{\mathcal{P}}$. At this stage, the robot identifies the set of frontier cells that are visible from its current cell. Frontier cells that are immediately adjacent to each other and share at least one Cartesian coordinate are then grouped and the center of the set represents a candidate frontier cell. A trajectory linking the robot's current cell and a frontier candidate is called a candidate trajectory.

The objective of the frontier-based exploration stage is to find the candidate within the frontier cells to which a candidate trajectory not only minimizes the distance to be traversed by the robot, but also is most "promising" with respect to fulfilling ϕ. Therefore, we propose an ordering relation of the frontier candidates based on the trajectory length to the frontier found by using a search algorithm, such as A^* [21], and the monitor's output of the prefix generated by such a trajectory. In other words, we assign a weight $w(i)$ for every frontier candidate i, defined as:

$$w(i) = m(\pi_i) \cdot \exp(-\gamma \cdot l(\tau_i)), \tag{1}$$

where $m(u_i)$ is the output of the monitor describing the future of the prefix π_i generated by the candidate trajectory τ_i and $l(\tau_i)$ expresses the trajectory length. The parameter γ expresses the importance of the path length over the future

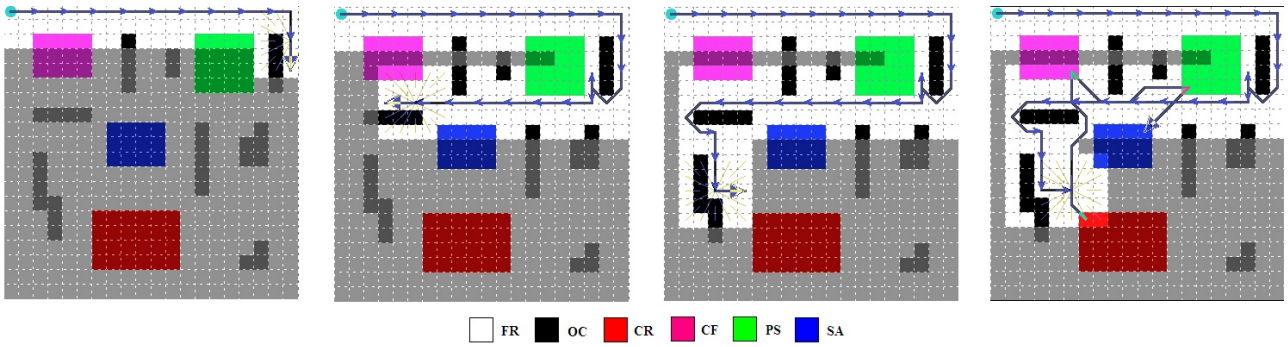

Fig. 2. Snapshots (to be read left-to-right) from a movie showing the robot's motion produced by applying our path planning algorithm to satisfy *Specification 1*. The robot is represented as a blue triangle and the arrows coming from the triangle represent the robot's sensor measurements.

aspect of the trajectory. Eq. (1) allows us to establish a trade-off between the cost of reaching a frontier cell and the utility of reaching that frontier cell. Moreover, Eq. (1) allows the robot to choose a more distant frontier if the candidate trajectory to that frontier is more promising in terms of satisfying ϕ.

For instance, consider Fig. 1 and the specification given as $\neg Obstacles\ \mathcal{U}\ Goal$. The robot's knowledge in Fig. 1.b is not sufficient to obtain a trajectory satisfying the specification. In the absence of occlusion by obstacles, the robot chooses a cell from the frontier candidates (in red) that allows it to obtain more information about the environment without falsifying the specification. This is depicted in Fig. 1.c, which shows the system after the robot moves to the frontier cell selected in the previous round and augments its knowledge of the environment. The process is repeated until the robot acquires the information needed to satisfy the task specification (Fig. 1.d). Note that one can also incorporate the appropriate approach to reason about occlusion into the planning process.

To compute the future aspect of the prefix generated by the candidate trajectory τ, we do a breadth first search on the automaton graph of the FSM $\mathcal{M}$ representing the monitor of ϕ to obtain the distances d_{bad}, d_{good} to the closest bad state $q_\perp$ (i.e. $\lambda(q_\perp) = \perp$) and the closest good state $q_\top$ (i.e. $\lambda(q_\top) = \top$) . These distances are then used to define the labeling function $\lambda_\mathcal{F} : Q \to \mathbb{Z}$ [22],

$$\lambda_\mathcal{F}(q) = d_{bad}(q) - d_{good}(q),$$

where d_{bad} and d_{good} evaluate to ∞ if no such state is reachable.

In order to project the computed aspect into the satisfaction value domain $[0, 1]$, let $\xi : \mathbb{N} \to [0, 1]$ be a strictly monotonic function. The future aspect of a trajectory τ generating a prefix π is given by:

$$m(\pi) = \xi(\exp(f(\pi))), \tag{2}$$

where $\xi(x)$ is chosen to be equal to $1 - \frac{1}{1-x}$, and $f(\pi)$ is the difference between the shortest bad prefix and the shortest good prefix for ϕ that is an extension of π. In the case the candidate trajectory violates ϕ (i.e., $m(\cdot) = 0$), the candidate frontier is discarded, and one of the cells adjacent to it within its region is chosen as the new candidate. Finally, the frontier candidate maximizing Eq. (1) is selected and its candidate trajectory is traversed.

D. Soundness and Completeness

The soundness and completeness properties of the algorithm are summarized in the following theorem.

Theorem 1. *Given a mobile robot deployed in an unknown partitioned environment with a known set of labels capturing certain elements of interest at its regions, and a specification given as an scLTL formula over these labels, the proposed algorithm returns a solution to Problem 1.*

Proof. Consider that an empty run resulted from seeking a satisfying trajectory in the product automaton, i.e., the formula cannot be yet satisfied. Let π be the word generated by a candidate trajectory. If π violates the specification, the monitor function evaluates to 0 discarding this trajectory. The algorithm then prevents the formula from being violated by selecting a trajectory that does not falsify the specification.

Each time the transition system is updated, the algorithm updates the corresponding product automaton. If there exists a path concatenating the robot's traversed path and an accepting state of the product automaton, the IBFS is guaranteed to find this path.

The algorithm terminates when an accepting run has been found or when there are no frontier cells left; i.e., the environment has been completely explored and yet no accepting run has been found in the product automaton after the most recent update of the transition system, or all paths to frontiers violate the specification. □

V. IMPLEMENTATION AND CASE STUDY

In this section, we first describe the software implementing our algorithm. Then, we show the results of our simulation when applying the algorithm.

A. Software and Simulation Implementation

The controller presented in Alg. 1 is implemented in the Matlab environment. In our implementation, we use scheck [23] to obtain the DFA $\mathcal{A}$ corresponding to the scLTL formula ϕ. Furthermore, the FSM $\mathcal{M}$ representing the monitor of ϕ is obtained by a slight modification of

the LTL_3 [24] tool. Our incremental automaton construction uses a modified version of the IBFS code [25].

Our simulated environment comprises a 2-D workspace partitioned into 400 cells and a mobile robot provided with an accurate radial laser range sensor of three cell range. Within the environment, we consider four areas of interest denoted by the following labels: *Safe* (**SA**), *Critical Functional* (**CF**), *Critical* (**CR**), and *Power Supply* (**PS**). *Critical* and *Critical Functional* are terms used in search and rescue operations to assess possible locations of victims, or areas of entrapment. *Critical* areas are characterized for the lack of entrance to the areas, while *Critical Functional* areas represent limited entrance to them. Furthermore, we distinguish the occupancy condition of each cell in the environment by means of the labels *Free* (**FR**), and *Occupied* (**OC**).

B. Simulation Results

Consider the environment shown in Fig. 2 and the following motion specification.

Specification 1. *"Visit a Critical area, then a Critical Functional area, and then a Power Supply area and, finally, go to a Safe region while avoiding any Critical and Critical Functional areas. Always avoid obstacles".*

This specification can be translated to the scLTL formula

$$\phi \;::\; \mathbf{FR}\;\mathcal{U}\;(\mathbf{CR} \wedge ((\mathbf{FR} \vee \mathbf{CR})\;\mathcal{U}\;(\mathbf{CF} \wedge ((\mathbf{FR} \vee \mathbf{CF})\;\mathcal{U}\;(\mathbf{PS} \wedge (\neg\mathbf{OC} \wedge \neg\mathbf{CR} \wedge \neg\mathbf{CF})\;\mathcal{U}\;\mathbf{SA}))))). \quad (3)$$

Fig. 2 highlights scenes from the simulation of our path planning algorithm for specification 1. For simplicity of presentation, the effects of occlusion by obstacles are ignored in this example. The black line corresponds to the actual trajectory executed by the robot. The robot begins its motion planning at the initial cell which is depicted with a cyan dot. Next the robot starts the exploration process by discovering new areas of the environment. Due to its limited sensing, the robot only senses part of the environment and some labels corresponding to the regions it sees. Nevertheless, note that the specification is never falsified. Once the robot has accumulated enough information to complete its task, it stops exploring and executes the trajectory obtained through the incremental automaton-based search. Such a trajectory is found by using the transition system integrating the acquired knowledge of the environment up to then. We selected γ in Eq. (1) to be equal to 0.3. The running time of this example was 6.78 seconds on a computer with 2.5 GHz dual processor.

VI. CONCLUSIONS

This paper presented a complete framework to solve the motion planning problem for a robot deployed in an unknown environment with a mission expressed as an scLTL formula. An algorithm was derived to find a trajectory that does not falsify the formula when the formula is yet to be satisfied given the current knowledge of the environment. The proposed algorithm exploits the existence of off-the-shelf model checking and runtime verification tools, the efficiency of graph search algorithms, and the efficacy of exploration techniques to solve the considered problem. To illustrate the effectiveness of our approach, we implemented the devised algorithm and applied it in a simulation setup.

REFERENCES

[1] M. Lahijanian, S. B. Andersson, and C. Belta, "Temporal logic motion planning and control with probabilistic satisfaction guarantees," *IEEE Trans. Robot.*, vol. 99, no. 6, pp. 1–14, 2011.

[2] A. I. Medina Ayala, S. B. Andersson, and C. Belta, "Probabilistic control from time-bounded temporal logic specifications in dynamic environments," in *Proc. IEEE Int. Conf. on Robot. and Autom.*, 2012, pp. 4705–4710.

[3] E. Plaku, "Planning in discrete and continuous spaces: From LTL tasks to robot motions," in *Towards Autonomous Robotic Syst.*, 2012, pp. 331–342.

[4] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proc. IEEE Conference on Decision and Control*, 2009, pp. 2222–2229.

[5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, Cambridge, Massachusetts, 2005.

[6] M. Y. Vardi and P. Wolper, "Probabilistic robotics," in *Proc. First IEEE Symposium on Logic in Computer Science*, 1986, pp. 332–344.

[7] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for LTL and TLTL," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, p. 14, 2011.

[8] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proc. 16th $ACMSIGPLAN-SIGACT$ Symposium on Principles of Programming Languages, ser. POPL 89*, 1989, pp. 179–190.

[9] S. Sarid, B. Xu, and H. Kress-Gazit, "Guaranteeing high-level behaviors while exploring partially known maps," in *Robotics: Science and Systems*, 2012.

[10] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *Proc. IEEE Int. Conf. on Robot. and Autom.*, 2012, pp. 5163–5170.

[11] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," *J. Comput. Syst. Sci.*, vol. 78, no. 3, pp. 911–938, 2012.

[12] A. Bauer, M. Leucker, and C. Schallhart, "The good, the bad, and the ugly, but how ugly is ugly?" in *Runtime Verification*, 2007, pp. 126–138.

[13] X. C. Ding, M. Lazar, and C. Belta, "Receding horizon temporal logic control for finite deterministic systems," *Compt. Research Repository*, vol. abs/1203.2860, 2012.

[14] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, no. 3, pp. 291–314, 2001.

[15] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *IEEE Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[16] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot planning and control in polygonal environments," *IEEE Trans. on Robotics*, vol. 21, no. 5, pp. 864–874, 2005.

[17] S. R. Lindemann and S. M. LaValle, "Simple and efficient algorithms for computing smooth, collision-free feedback laws over given cell decompositions," *International Journal of Robotics Research*, vol. 28, no. 5, pp. 600–621, 2009.

[18] A. V. Goldberg, S. Hed, H. Kaplan, R. E. Tarjan, and R. F. F. Werneck, "Maximum flows by incremental breadth-first search," in *ESA*, 2011, pp. 457–468.

[19] B. Yamauchi, "Frontier-based exploration using multiple robots," in *Agents*, 1998, pp. 47–53.

[20] S. Koenig, M. Likhachev, Y. Liu, and D. Furcy, "Incremental heuristic search in AI," *AI Magazine*, vol. 25, no. 2, pp. 99–112, 2004.

[21] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, Third Edition*. Prentice Hall, 2009.

[22] N. Decker, "A continuous truth domain for runtime verification," University of Stuttgart, Institute of Formal Methods in Computer Science, Theoretical Computer Science, Tech. Rep., 2011.

[23] T. Latvala, "Efficient model checking of safety properties," in *10th Int. SPIN Workshop*, 2003, pp. 74–88.

[24] A. Bauer, "LTL_3 tools." [Online]. Available: http://ltl3tools.sourceforge.net/index.html

[25] "Incremental BFS code." [Online]. Available: http://www.cs.tau.ac.il/ sagihed/ibfs/