Semantic Abstraction-Guided Motion Planning for scLTL Missions in Unknown Environments

Author Names Omitted for Anonymous Review.

***Abstract*—Complex mission specifications can be often specified through temporal logics, such as Linear Temporal Logic and its syntactically co-safe fragment, scLTL. Finding trajectories that satisfy such specifications becomes hard if the robot is to fulfil the mission in an initially unknown environment, where neither locations of regions or objects of interest in the environment nor the obstacle space are known a priori. We propose an algorithm that, while exploring the environment, learns important semantic dependencies in the form of a semantic abstraction, and uses it to bias the growth of an RRG graph towards faster mission completion. Our approach leads to finding trajectories that are much shorter than those found by the sequential approach, which first explores and then plans. Simulations comparing our solution to the sequential approach, carried out in 100 randomized office-like environments, show more than 50% reduction in the trajectory length.**

I. Introduction

Motion planning with Linear Temporal Logic (LTL) mission specifications aims for consideration of richer objectives than the traditional A-to-B motion planning. Examples of such objectives include periodic surveillance, request-response, or sequencing. Successful approaches to the problem range from using various cell decomposition techniques, to creating roadmaps abstracting the environment and to sampling-based motion planning. Motion planning with LTL missions is, however, much more challenging in *a priori* unknown environments: efficient treatment of LTL specifications may require exploiting semantic and spatio-temporal dependencies between features of the environment, which are typically unknown beforehand. As an example, consider that we would like a robot to check all waste bins in all offices in an office environment. When finding the first bin, the robot may realize it was next to a desk. While looking for the bin in the next office, it is most natural that the robot starts exploring again next to the desk. At the same time, due to the potential complexity of the environment, it is not desirable to stick fully to all of the observed semantic and spatio-temporal correlations as not all of them are relevant for the specification satisfaction.

In this paper, we focus on sampling-based motion planning with missions specified with the syntactically co-safe fragment of LTL (scLTL), and with the robot deployed in *a priori* unknown environments. The key idea of our approach is, on the conceptual level, to make the sampling *guided* by a semantic abstraction of the system and by the specification. The overview of our algorithm is depicted in Fig. 1. We extend the RRG algorithm with learning and biasing; we iteratively learn a semantic abstraction of the system from the gradually growing RRG graph and compose it with an automaton representation of the specification into a so-called product. The product is used to bias sampling in RRG, i.e. to exploit the semantic and spatio-temporal dependencies of features in the environment as well as their relation to satisfying the desired specification.

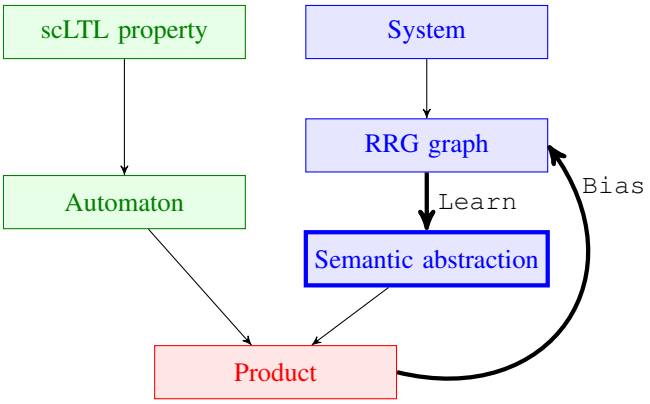

Figure 1. Scheme of our model-checking-inspired approach with novel elements drawn thickly.

Compared to the naive two-step approach, which first explores the environment and then plans a trajectory that satisfies the mission, our approach (i) performs both tasks at once and, moreover, (ii) allows mutual exchange of information between the two tasks. We show that these two improvements shorten the length of the executed path significantly. We achieve this while maintaining similar computation time, which will, in reality, be negligible as the robot can execute the algorithm in real-time while navigating in the environment. Our contribution can be summarized as follows:

- We propose a method to learn a semantic abstraction of the system, suitable for planning with scLTL missions.
- We exploit the learned semantic abstraction and, together with consideration of the specification, we bias the growth of the RRG graph towards promising regions (in terms of making progress towards the specification satisfaction).
- We experimentally show that the loop between sampling and learning leads to better planning in terms of shorter trajectories when compared to the naive two-step approach. The results indicate more than 50% savings.

The paper is organized as follows. Sec. I-A introduces relevant related work, and Sec. II describes preliminary tools needed for the remainder of the paper. The problem is formally defined in Sec. III, which is followed by the proposed solution

and analysis in Sec. IV. Lastly, a case study is presented in Sec. V, with conclusions and future work in Sec. VI.

A. Related Work

One of the first works to propose the use of a sampling-based motion planning algorithm to find a trajectory that satisfies a temporal logic specification is [9]. In that work, the authors propose the Rapidly-exploring Random Graph (RRG) as an alternative to the Rapidly-exploring Random Tree (RRT) to finding cyclic trajectories that satisfy a deterministic μ-calculus specification. Another approach is presented in [4], but this time for the syntactically co-safe fragment of Linear Temporal Logic (scLTL). Following these, Vasile and Belta [17] propose improvements to [9], more specifically for dealing with full LTL and for improving scalability. None of these works, however, deals with partially-known environments, nor do they attempt to speed up the search by learning characteristics of the environment.

More recently, Kantaros and Zavlanos [6] described an approach for multi-robot systems under global temporal tasks. Instead of using an RRG, the authors propose a two-step approach using RRT*. The first step constructs a tree until an accepting state of the automaton capturing the evolution of the LTL formula is reached. The second step then grows another tree rooted at this accepting state, and attempts to find a cyclic (infinite) path that satisfies the LTL specification. The same authors then introduce in [7] sampling bias guided by the automaton capturing the LTL, something that [13] also proposes in a similar fashion. Lastly, besides proposing a heuristic to guide the search, [16] integrates feedback control laws to guarantee feasibility of plans by robots with complex, possibly non-holonomic, dynamics. Although these works propose ways of improving the time taken to find a plan, they all rely on having details of the environment a priori.

To the best of our knowledge, the two papers that are mostly related to ours are [8] and [1]. The former proposes a reactive sampling-based algorithm for path planning in unknown environments under scLTL specifications. However, differently from what we propose, only the obstacle space is initially unknown to [8], i.e. the locations of the regions of interest, therefore the labeling function, are known a priori. On the other hand, Ayala et al. [1] considers completely unknown environments, including the labeling function. However, the authors propose an approach over a discretized partitioning of the environment, performing frontier exploration [18] until a path that satisfies the scLTL specification is found. We merge benefits of both approaches by proposing a sampling-based approach on completely unknown environments; furthermore, we propose a way of learning relations between labels, together with exploiting them for guiding the path search.

When it comes to robotic deployment in unknown environments, a crucial initial step might be to efficiently create a map in an exploratory manner. A seminal work on exploration is by Yamauchi [18], in which the author proposes the method coined *frontier exploration*. Since then, several other approaches have been proposed. Among them is the Receding Horizon Next-Best-View Planner [5] and the Autonomous Exploration Planner [15], both building upon RRT*. These works, however, do not focus on capturing various dependencies and relations in the environment. In contrast, in a probabilistic approach proposed by Aydemir et al. [2], a robot uses common-sense knowledge about the relation between objects and semantic room categories. Here, the focus is however on search for objects and not satisfaction of complex LTL goals.

II. Preliminaries

Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{R}^n$ the n-dimensional Euclidean space. We use Σ for the finite set of atomic propositions. For a set X, 2^X denotes its power set. A word over an alphabet Y is a sequence of elements of Y. The exclusive-or operation is denoted by $\oplus$, and the disjoint union of sets by $\uplus$.

Consider a robot deployed in an *environment* $\mathcal{X} \subset \mathbb{R}^n$ and let $x_0 \in \mathcal{X}$ be its initial state. Let $\{\mathcal{O}_1, \mathcal{O}_2, \dots \mathcal{O}_k\}$ be the set of obstacles such that $\mathcal{O}_i \subset \mathcal{X}$ for all $i \in [1, k]$, and $\mathcal{X}_{\text{free}} = \mathcal{X} \setminus \bigcup_{i=1}^{k} \mathcal{O}_i$ denotes the obstacle-free space. A trajectory in the environment $\mathcal{X}$ is defined by a continuous function $\sigma : [0, 1] \to \mathcal{X}$. A trajectory is collision-free if $\sigma(t) \in \mathcal{X}_{\text{free}}$, $\forall t \in [0, 1]$. Regions of the environment $\mathcal{X}$ are labelled with atomic propositions Σ according to a labelling function $L : \mathcal{X} \to 2^\Sigma$, which maps each state in the state-space to a set of atomic propositions that hold true there.

A map of the environment $\mathcal{X}$ is a partitioning into a finite number of cells of equal size with a predefined precision, which can be labeled as *free*, meaning that the cell lies in $\mathcal{X}_{\text{free}}$, *occupied*, if any point within the cell lies inside the obstacle space (corresponding to an over-approximation of the obstacle set), or *unmapped*, that highlights the cell has not been seen by the robot so far. Every cell is initialised as *unmapped*, and is updated whenever it lies in the line-of-sight of the robot. A cell is called a *frontier* cell if it is marked as *free* and has a neighbouring cell marked as *unmapped*. A *map frontier* is a connected group of frontier cells, and its size is its cardinality. This is a common approach among the 3D-exploration community, so we refer to papers such as [18, 5] for more details.

A. RRG

The Rapidly-exploring Random Graph [10] is an anytime[1] sampling-based motion planning algorithm that builds a connected roadmap. It incrementally builds a graph $G = (V, E)$ such that $v \in \mathcal{X}_{\text{free}}$, $\forall v \in V$, and an edge $e \in E$ connects two nodes $v_a, v_b \in V$ if there exists a collision-free trajectory $\sigma_{v_a}^{v_b}$ between them, with $\sigma_{v_a}^{v_b}(0) = v_a$ and $\sigma_{v_a}^{v_b}(1) = v_b$. A path over G is a sequence of nodes $p = v_0, v_1, v_2, \dots$ such that $v_i \in V$ and $(v_i, v_{i+1}) \in E$, for all $i \geq 0$.

[1] An anytime algorithm returns a valid solution even if it is interrupted before termination; moreover, the longer it runs, the more its solution is improved.

B. Syntactically co-safe LTL and DFA

Definition 1 ((Syntactically co-safe) Linear Temporal Logic [14, 11])**.** *A formula of* LTL *is given by the syntax:*

$$\varphi := a \mid \neg a \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2 \mid \mathbf{G}\varphi$$

where, $a \in \Sigma$ is an atomic proposition, $\neg, \wedge, \vee$ are the Boolean operators 'negation', 'conjunction', and 'disjunction', respectively. $\mathbf{X}$*,* $\mathbf{U}$*,* $\mathbf{G}$ *denote the LTL operators 'next', 'until', and 'globally' respectively. The* syntactically co-safe fragment of LTL (scLTL) *is given by the same syntax, but prohibiting the operator* $\mathbf{G}$.

The semantics of LTL formulas is defined on words over 2^Σ. The Boolean operators have usual semantics. Intuitively, $\mathbf{X}\varphi$ means that φ is true in the next time step and $\varphi_1 \mathbf{U} \varphi_2$ asserts that φ_1 will be true until φ_2 becomes true. $\mathbf{F}$ is known as the 'finally' or 'eventually' operator whose semantics asserts that the property φ becomes true at some point in the future. As such, it can be defined in terms of $\mathbf{U}$ as $\mathbf{F}\ \varphi \equiv \mathit{true}\ \mathbf{U}\ \varphi$ $\mathbf{G}$ is known as the 'globally' or 'always' operator with the semantics that φ is always satisfied. Since the robot moves in continuous time and $\mathbf{X}$ operator is usually defined for discrete time steps, we consider for simplicity properties without $\mathbf{X}$. However, our approach is applicable for the whole of LTL.

Let $\mathcal{L}(\varphi)$ denote the set of words that satisfies the LTL formula φ.

Definition 2 (Deterministic Finite Automaton)**.** *A* deterministic finite automaton (DFA) *is a tuple $(2^\Sigma, Q, q_0, \delta, F)$ where 2^Σ is the alphabet, Q is a finite set of states, q_0 is an initial state, $\delta : Q \times 2^\Sigma \to Q$ is a transition function and $F \subseteq Q$ is the set of accepting states.*

A run over a word $w_1, \ldots, w_n$ is a sequence of states $q_0, q_1, \ldots, q_n$ such that $q_i = \delta(q_{i-1}, w_i)$ for all i. A word is accepted by the automaton if the run over the word end in F. We define the language accepted by an DFA $\mathcal{A}$ as $\mathcal{L}(\mathcal{A}) = \{w \in (2^\Sigma)^\omega \mid w \text{ is accepted by } \mathcal{A}\}$. It is a standard result that for every scLTL formula φ, there exists a DFA $\mathcal{A}$ such that $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$ and it is effectively constructible. Consequently, DFA can be used as a precise representation of an scLTL property. (Our approach can be extended in a straightforward way to so-called Büchi automata, which can express the whole of LTL.)

Definition 3 (State-labelled transition system)**.** *A (state-labelled) transition system (TS) is a tuple (S, s_0, Δ, L) where S is a finite set of states, s_0 is an initial state, $\Delta : S \to 2^S$ is a transition relation, and $L : S \to 2^\Sigma$ is the labelling function.*

A transition system (representing the real system or its abstraction) can be combined with an automaton (representing the property) into a *product*, see Fig. 1. Runs of the product are thus runs of the transition system monitored by the automaton. The automaton always reads the atomic propositions true in the current state and, on the whole, determines whether the run satisfies the property or not. This standard construction is often used in model checking [3] and we use it to improve RRG by *mutual* exchange of information between the two parts.

Definition 4 (Product)**.** *Given a TS $\mathcal{T} = (S, s_0, \Delta, L)$ and DFA $\mathcal{A} = (2^\Sigma, Q, q_0, \delta, F)$, the product $\mathcal{T} \times \mathcal{A}$ is the tuple $(S \times Q, \hat{s_0}, \hat{\Delta}, \hat{F})$ where*

- $\hat{s_0} = (s_0, \delta(q_0, s_0))$,
- $\hat{\Delta}((s, q)) = \{(s', \delta(q, s')) \mid s' \in \Delta(s)\}$,
- $\hat{F} = \{(s, q) \mid q \in F\}$.

III. Problem Formulation

Consider a robot deployed in an *a priori* unknown environment. We assume that the set of atomic propositions Σ (semantic labels, such as $living\_room$ or $wastebin$) is known beforehand, but not where they hold. In other words, the L function, as well as the obstacle-space, are unknown. Furthermore, we also assume that the robot is equipped with adequate sensors and perception modules that can identify labels and obstacles within a sensing radius r_s around its current position.

Problem 1. *Given an initial state $x_0 \in \mathcal{X}$ in an a priori unknown environment $\mathcal{X}$, and an scLTL specification Φ over the set of atomic propositions Σ, find a collision-free trajectory σ in $\mathcal{X}_{free}$ which satisfies Φ.*

Since neither obstacles nor the labeling function are known a priori, one cannot use traditional offline approaches described in Sec. I-A to solve Problem 1. The solution must be an online algorithm that learns the obstacle space and the labeling function as it moves in the environment. A straightforward way to solve this problem would be to explore the whole environment and assign labels to features in the environment first, and then use planning approaches. We propose to integrate exploration and planning. As a result, the robot attempts to make progress towards satisfying the specification while exploring, resulting in a possibly shorter travelled distance.

IV. Solution

Our solution to Problem 1 is an algorithm that learns interesting semantic dependencies and relations in the form of a *semantic abstraction* and utilizes this knowledge to bias the growth of a motion graph towards faster satisfaction of the desired LTL specification.

A. Semantic abstraction-guided RRG

The overall Semantic abstraction-guided RRG (SAG-RRG) procedure is overviewed in Alg. 1. Similarly to RRG, the procedure builds a graph $G = (V, E)$ whose vertices $v \in V$ lay within the obstacle-free space $\mathcal{X}_{\text{free}}$, and edges $e \in E$ connect two vertices if a collision-free trajectory exists. An iteration of the algorithm starts by updating the map of the environment with information of what is within the sensing radius r_s of the robot (line 5). After that, it computes the guidance according to the *Bias* function (line 6), which is described in more detail in Alg. 3 and Sec. IV-C. Then, each iteration of the internal *while* loop (lines 8-24) attempts to add one new vertex to G, in a similar way to the RRG algorithm. It

samples a point from the known-space of the environment and finds its closest neighbour in the current graph (line 9). If the path connecting these two points is collision-free, the sample is considered for being added to the graph. If the symbolic counterpart of the sampled transition is in bias, the sampled point is stored as a "bias frontier"; otherwise it rejects this sample with some probability p (lines 11-14). This probability depends on how much you want to bias the sampling. The algorithm then follows the usual RRG procedure: it adds the new vertex and edge to the graph (line 15) and attempts to connect such vertex to its closest neighbours (lines 19-24), with slight modifications for checking for bias frontiers, and for keeping track of the symbolic transitions t_{symb} (lines 16 and 22) and states seen seen_st (line 17). After sampling a batch, it updates the semantic abstraction through the *Learn* procedure (line 25), which is detailed in Alg. 4 and in Sec. IV-B. The algorithm then calls the *Move* procedure (Alg. 4 and Sec. IV-D), which finds the best frontier to move to, and moves the robot to the point in G closest to it. Finally, the procedure checks if a plan that satisfies the LTL formula has been found.

Remark 1. In Alg. 1 an edge $e \in E$ is defined in a way to ensure the labels along it change only once. Formally, given an edge $e = (v_a, v_b) \in E$, there exists a state $x \in \sigma^{v_b}_{v_a}$ such that i) $L(x') = L(v_a)$, $\forall x' \in \sigma^{x}_{v_a}$, and ii) $L(x'') = L(v_b)$, $\forall x'' \in \sigma^{v_b}_{x+\epsilon}$, where $x + \epsilon$ represents a state in the neighbourhood of x.

Algorithm 1: SAG-RRG

Input: $\mathcal{X}, x_0, \Phi$
Output: A collision free trajectory in $\mathcal{X}$ which satisfies Φ

1 Initialize semantic abstraction
2 $V \leftarrow x_0$; $E \leftarrow \emptyset$
3 curr_pos $\leftarrow x_0$; seen_st $\leftarrow s(x_0)$
4 **while** $\neg$`AcceptingPath()` **do**
5 `UpdateMap(`curr_pos$, r_s$`)`
6 $bias \leftarrow$ `Bias(`seen_st`)`
7 $t_{\text{symb}} \leftarrow \emptyset$; $i \leftarrow 0$
8 **while** $i <$ batch_size **do**
9 $[x_s, x_{near}] \leftarrow$ `SampleAndExtend(`$\mathcal{X}_{free}, V$`)`
10 **if** `CollisionFree(`x_{near}, x_s`)` **then**
11 **if** $(s(x_{near}), s(x_s)) \in bias$ **then**
12 add x_s to bias frontiers
13 **else**
14 **continue** to next iteration with prob p
15 $E \leftarrow E \cup (x_{near}, x_s)$; $V \leftarrow V \cup x_s$
16 $t_{\text{symb}} \leftarrow t_{\text{symb}} \cup (s(x_{near}), s(x_s))$
17 seen_st $\leftarrow$ seen_st $\cup\, s(x_s)$
18 $i \leftarrow i + 1$
19 **for** $x \in$ `Near(`x_s`)` **do**
20 **if** `CollisionFree(`x, x_s`)` **then**
21 $E \leftarrow E \cup (x, x_s)$; $V \leftarrow V \cup x$
22 $t_{\text{symb}} \leftarrow t_{\text{symb}} \cup (s(x), s(x_s))$
23 **if** $(s(x), s(x_s)) \in bias$ **then**
24 add x_s to bias frontiers
25 `Learn(`t_{symb}`)`
26 curr_pos $\leftarrow$ `Move()`
27 **return** *accepting_path*

B. Learn

This section describes in detail the proposed approach to learning the semantic abstraction of the environment. Intuitively, we try to find transitions that are similar to the sampled ones and add them as special, potential transitions in the abstraction. Next part describes how we can accommodate these special transitions in the abstraction.

1) Semantic Abstraction: To formalize the semantic abstraction, we propose extending the state-labelled TS to a "multi-modal" transition system, our extension of modal transition systems [12]:

Definition 5 (Multi-Modal Transition System)**.** *A tuple $(S, s_0, \Delta, L, \mathbb{M}, \mathcal{M})$ is called a multi-modal transition system (MM-TS), where (S, s_0, Δ, L) is a state-labelled transition system (Def. 3), $\mathbb{M}$ is a finite ordered set of modes, and $\mathcal{M} : \Delta \to \mathbb{M}$ is a modal marking.*

A semantic abstraction of an RRG graph is an MM-TS, where a discrete state $s \in S$ represents a set of points $x \in \mathcal{X}$ with the same labeling. With a slight abuse of notation, we use $x \in s$ to say that $L(x) = L(s)$ and $s(x)$ to denote $s \in S$, such that $x \in s$.

Intuitively, $\mathcal{M}$ assigns to each transition in the abstraction a "degree" of confidence that a corresponding transitions is present in the corresponding concrete points. We use two modes[2] in our MM-TS: *must* and *may*. The former is used for transitions that are known to exist based on samples taken from the environment while the graph is constructed; the latter is an extrapolation to which transitions might exist based on the *must* transitions. When a new edge (x, x_{new}) is added to the SAG-RRG graph G, a transition $(s(x), s(x_{new}))$ is added to the MM-TS as a *must* transition, and similar transitions (see Def. 7) are added as *may* transitions. Let us now formalize when we deem two transitions of a MM-TS *similar*.

Definition 6 (Domain of Change)**.** *The domain of change for a transition $(s, s') \in \Delta$ is $DoC(s, s') = L(s) \oplus L(s')$.*

The *domain of change* is essentially the set of all atomic propositions which changed their valuation during the corresponding transition in the MM-TS. For example, given a transition (s, s') where $L(s) = \{a, b\}$ and $L(s') = \{b, c\}$, its $DoC(s, s')$ is $\{a, c\}$.

Definition 7 (Similar Transitions)**.** *Two transitions $(s, s'), (\bar{s}, \bar{s}') \in \Delta$ are* similar *if and only if $DoC(s, s') = DoC(\bar{s}, \bar{s}')$, and $\forall a \in DoC(s, s')$, $a \in L(s) \iff a \in L(\bar{s})$ and $a \in L(s') \iff a \in L(\bar{s}')$.*

[2]Although we choose to use two modes in this paper for the sake of simplicity of the exposition, the approach presented throughout the paper is generic enough to use any number of modes. Besides *must* and *may*, one could also use *may not* and *must not*, for instance.

Algorithm 2: Learn

```
1 Function Learn(t_symb):
2     AddToProduct(t_symb, must)
3     t_sim ← FindSimilar(t_symb)
4     AddToProduct(t_sim, may)
```

Intuitively, similar transitions behave the same on their *domain of changes*. For example, a transition (s, s'), where $L(s) = \{a, b\}$ and $L(s') = \{b, c\}$, is similar to $(\bar{s}, \bar{s}')$ where $L(\bar{s}) = \{a, d\}$ and $L(\bar{s}') = \{d, c\}$. The idea is that after experiencing the transition (s, s') which leaves b untouched, we may hypothesize b is irrelevant and that the same behaviour is present also in the situation when b does not hold and when some other irrelevant proposition, e.g. d, holds. However, b still may be a precondition for the transition, hence we introduce the new transition $(\bar{s}, \bar{s}')$ only with a low "confidence".

The formal definition of similarity allows us to clearly identify when transitions in the MM-TS, i.e. the semantic abstraction of an RRG graph, should be marked as *may*.

2) Multi-modal product: The semantic abstraction captures existing and possible dependencies and relationships between labels in the environment regardless of the desired specification. We extend the definition of product (Def. 4) to incorporate the knowledge of the specification and thus enable biasing of SAG-RRG sampling to achieve faster specification satisfaction. In short, a multi-modal product (MM-P) is a product as in Def. 4 but with a MM-TS instead of a TS.

Definition 8 (Multi-modal Product)**.** *Given a MM-TS $(S, s_0, \Delta, L, \mathbb{M}, \mathcal{M})$ and a DFA $\mathcal{A} = (2^{\Sigma}, Q, q_0, \delta, F)$, their product (MM-P) is a tuple $(S \times Q, \hat{s_0}, \hat{\Delta}, \hat{F}, \mathbb{M}, \hat{\mathcal{M}})$, where the first four components are defined as in Def. 4 and the remaining two are the modes $\mathbb{M}$ and a model marking $\hat{\mathcal{M}} : \hat{\Delta} \to \mathbb{M}$, such that $\hat{\mathcal{M}}((s, q), (s', q')) = \mathcal{M}(s, s')$.*

Similarly to the multi-modal transition system, the product can be constructed iteratively, along with the construction of the SAG-RRG graph.

3) Learn procedure: The procedure Learn is summarized in Alg. 4. Given a set of transitions t_{symb}, this procedure adds them to the MM-TS as *must* transitions, since we know that these transitions are already there. After that, for each $t \in t_{symb}$, it computes the transitions similar to t and add them as *may* transitions in the MM-TS.

C. Bias

The *bias* procedure computes which transitions would more quickly bring the system to an accepting state of the LDBA. It returns a hierarchical list of transitions according to how far they are from an accepting state in MM-P; the closer a transition is to an accepting state, the better. These transitions can then be used to bias the construction of the motion graph for faster convergence.

The procedure, described in Alg. 3, starts by initializing the variables $bias$ and reached, which store transitions and states, respectively. The first element of $bias$ is the set of all transitions ending in accepting states of the MM-P (line 2). As for reached, it keeps track of all backwards-reachable states from the accepting states; hence its first element is the set of accepting states (line 3), and the second element is the pre-image of the accepting states (line 4). Then, until all the backward-reachable sets have been considered, $bias$ is constructed iteratively based of the set of sampled states seen_st (lines 7-13). In the end, ith element of $bias$ will be the set of states that can reach an accepting state after exactly i steps in the MM-P.

Learn and Bias functions work in unison and help each other improve. The more *may* transitions are learned, the better is the bias received. The better the bias, the more new transitions are learned and the faster it converges.

Algorithm 3: Bias

```
1  Function Bias(seen_st):
2      bias[0] ← transitions ending in accepting states acc_st
3      reached[0] ← acc_st
4      reached[1] ← PreImg(acc_st)
5      all_reached ← reached[0] ∪ reached[1]
6      i ← 1
7      while PreImg(reached[i]) ⊈ all_reached do
8          useful_pre ← PreImg(reached[i]) ∩ seen_st
9          useful_post ← PostImg(useful_pre) ∩ reached[i]
10         bias[i] ← (useful_pre, useful_post)
11         reached[i + 1] ← PreImg(reached[i])
12         all_reached ← all_reached ∪ reached[i + 1]
13         i ← i + 1
14     return bias
```

Algorithm 4: Move

```
1 Function Move:
2     p_1 ← FindBestMapFrontier()
3     p_2 ← FindBestBiasFrontier()
4     return Best(p_1, p_2)
```

D. Move procedure

The idea behind the Move procedure, described in Alg. 4, is to decide where to move next: should we go towards a place that will provide more information about the map, or should we move according to the advice that has been given by Bias? In order to compare both options, we employ the concept of *information gain* (IG). Given a map frontier, its information gain is defined as $\mathrm{IG}_{\mathrm{map}} = \mathrm{size} \times f(d)$, where size is the size of the frontier and $f(d)$ is a strictly decreasing function (for $d > 0$) of the distance from the robot to the center of the frontier.

In a similar fashion, we define the information gain of a *bias* frontier. Note that *bias* frontiers were introduced in Alg. 1 (lines 12 and 24) as a means to keep track of the vertices in V that correspond to advices given by Alg. 3. Since $bias$ is a list of transitions, we can associate a *rank* r with each transition from $bias$ equal to $\mathrm{index} + 1$, where index is the

index of the sampled transition. We define IG of these frontiers as $\text{IG}_{\text{bias}} = g(r, d)$, where g is some function such that both $g(r, \cdot)$ and $g(\cdot, d)$ are strictly decreasing, where d is again the distance from the robot to the frontier.

The intuition behind the IG of the *map* frontiers is to have a larger value the larger the frontier is, but penalise it according to its distance to the robot, so as to motivate exploration of smaller frontiers that are nearby. Similarly, with the IG of the *bias* frontiers, we want to motivate movement towards low-rank frontiers, since these are closer to satisfying the formula.

E. Analysis

Theorem 1. *The algorithm is sound, i.e. any trajectory returned by SAG-RRG satisfies the given scLTL formula* Φ.

Proof: (Sketch) The proposed algorithm iteratively constructs a product MM-P (Def. 4) between a semantic abstraction of the RRG graph and the automaton $\mathcal{A}$, which accepts exactly the language of the specification Φ. Paths in the product that visit accepting states project directly onto accepting runs of the automaton and runs of MM-TS, which in turn project directly onto paths in the RRG graph G and further onto trajectories of the robot in the workspace. Altogether, these trajectories necessarily satisfy Φ. ∎

Theorem 2. *SAG-RRG is asymptotically complete.*

Proof: (Sketch) Follows directly from the convergence and completeness properties of the original RRG [10] and the fact that the biasing we introduced allows to eventually sample the whole space. Regardless of the scenario, including the one with no regularity in the environment that can be learned and exploited for guiding the search, the worse-case scenario will see the proposed approach perform an exhaustive search of the environment. ∎

V. Implementation and Experiments

The proposed approach was implemented in Java and run on a consumer grade hardware (2.60GHz Intel i7-9750H CPU, 32 GB RAM). Binary Decision Diagrams (BDDs), which are very efficient for manipulating sets of Boolean variables, are used for storing and manipulating the product automaton, the labels of each node in the RRG, and the *bias*. We encode the RRG as an undirected graph whose nodes also store the labels that hold true at that state. We use the Java Spatial Index RTree library for spatial indexing and faster querying of nearest neighbours. JavaBDD and JGraphT are the libraries used for encoding the BDDs and the graph, respectively. We also use `jhoafparser` library to parse the automaton file.

An example of the office-like environment used for the case study is presented in Fig. 2. In order to draw statistically-meaningful results, 100 different instantiations of the environment were randomly generated, in which the footprint (i.e. walls and doors) of the office space remains unchanged, but desks and wastebins are randomly positioned within the rooms (without blocking the door).

The scLTL specification is inspired by a realistic scenario, common in every office environment: reach a wastebin in the

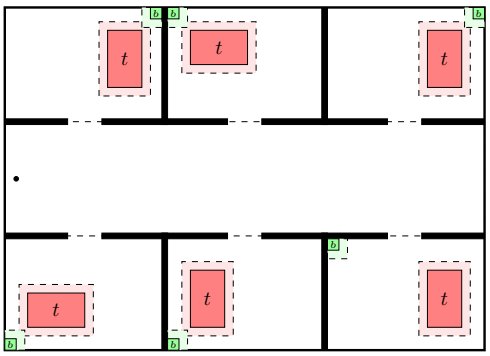

Figure 2. Example of an office-like environment used in the case study. Black solid lines represent walls. There are six rooms, each labeled with one atomic proposition r_i, for $i \in [1, 6]$, and a hallway labeled h. Each room contains a table (red) and a bin (green), labeled t and b, respectively. The labels of tables and bins hold true within the corresponding dashed and shaded areas surrounding it. The initial position of the robot is marked with a black dot on the left side of the hallway.

office rooms. We translate such a specification to the following scLTL formula:

$$\varphi = \mathbf{F}(r_1 \wedge b) \wedge \mathbf{F}(r_2 \wedge b) \wedge \ldots \mathbf{F}(r_6 \wedge b) \tag{1}$$

Note that such a specification does not impose any ordering of events.

For information gain, we use the following functions in our simulations

$$\text{IG}_{\text{map}}(m, p) = \frac{\text{size}_m}{d_{m,p}} \tag{2}$$

$$\text{IG}_{\text{bias}}(x_s, p) = \frac{a}{r^b_{x_s}\, d_{x_s,p}} \tag{3}$$

where size_m is the size of frontier m, $d_{m,p}$ and $d_{x_s,p}$ are the length of the shortest path between p and m and x_s, respectively, $a, b > 0$ are user-defined parameters, and r_{x_s} is the rank of x_s. Adjusting a, b is intuitive: i) suppose m and x_s are equidistant from p; ii) fix r_{x_s} to 1 and choose a to reflect how a *bias* frontier compares to a *map* frontier; iii) now suppose $x_{s,1}, x_{s,2}$ equidistant from p, such that $r_{x_{s,1}} = 1$ and $r_{x_{s,2}} = 2$; iv) choose b as to reflect how the importance of *bias* frontier decays with its rank (e.g. linearly, quadratic). For our case study, we chose $a = 100$ and $b = 2$.

The results are presented in Table I for 100 randomly-generated environments. The solution presented in this paper can be seen as an approach that performs exploration of the environment and planning to satisfy the scLTL mission concurrently, without or with bias in building the RRG tree ('Simultaneous' and 'Simult. biased' columns in Table I). We compare this integrated solution to the trivial sequential approach ('Explore, then plan' column in Table I), which consists of first exploring the whole environment, and then planning a trajectory that satisfies the mission. Lastly, in a more technical variant regarding the sensing capabilities of the robot, we analyse two cases, one where the robot can "see through" the desks (e.g., a flying robot), and another where they are considered to be Opaque. A few snapshots of the

Table I

MEAN AND STANDARD DEVIATION OF THE TOTAL TRAJECTORY LENGTH TO SATISFY THE SCLTL MISSION (1), ALONG WITH TOTAL RUNTIME AND RRG SIZE. THE RESULTS WERE DRAWN FROM SIMULATING EACH APPROACH 3 TIMES IN EACH OF THE 100 RANDOMLY-GENERATED ENVIRONMENTS.

	See-through Desks			**Opaque Desks**		
	Explore, then plan	**Simultaneous**	**Simult. biased**	**Explore, then plan**	**Simultaneous**	**Simult. biased**
Total length	77.3 (7.5)	56.6 (8.0)	29.4 (5.0)	79.1 (7.1)	62.9 (16.5)	32.3 (11.8)
Exploration length	57.1 (3.2)	37.5 (7.1)	28.0 (4.9)	57.8 (4.9)	44.4 (16.6)	31.3 (12.1)
Remaining plan l.	20.2 (7.0)	19.1 (3.6)	1.3 (1.8)	21.3 (5.1)	18.5 (3.4)	1.1 (1.8)
Total Time	7.8 (2.0)	6.4 (2.3)	7.3 (1.9)	9.6 (2.5)	8.3 (3.2)	9.1 (2.4)
RRG size	1931.2 (460.9)	1938.6 (559.5)	1793.6 (312.1)	2313.8 (550.9)	1868.7 (498.2)	1901.4 (301.2)

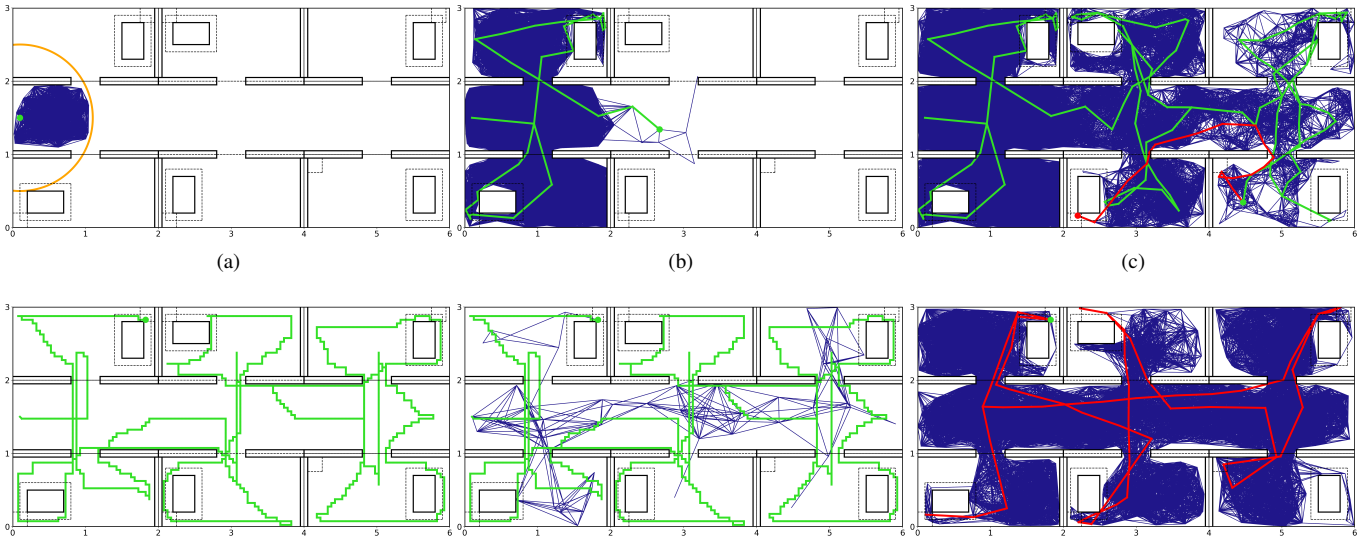

Figure 3. Snapshots of the robot navigating the office environment in the attempt to satisfy the scLTL mission (1) with two different approaches. The yellow semi-circle in (a) corresponds to the robot's sensing radius. The top-row figures (a-c) display the trajectory (green) when using the approach proposed by us (SAG-RRG), where exploration and planning are done together; the bottom-row figures (d-f) show the case where the robot first explores the environment, and only then it plans a path that satisfies the mission. The RRG graph at the time of the snapshot is shown in blue, and the path that leads to satisfaction of the mission is in red (c,f).

robot trajectory are shown in Fig. 3. Each approach is run three times in each of the 100 environments, totalling 1800 runs of the experiment.

The rows in Table I display the length ('Total length') of the trajectory traversed by the robot, from its initial state (common to all cases) until mission satisfaction, as well as the total computation time ('Total time') and number of nodes in the RRG graph ('RRG size'). Additionally, we also display the 'Exploration length' which, for the sequential approach, represents the length of the trajectory traversed only during the exploration phase, while for our approach it represents the length traversed until the system realises that a trajectory that satisfies the mission already exists. 'Remaining plan l.' is the length of the remaining trajectory that needs to be followed in order to satisfy the desired specification at the moment when exploration phase ends in the 'Explore, then plan' case, or the moment when the trajectory is found in the 'Simultaneous' and Simult. biased' cases. In Table I we see that having "see-through" desks makes the performance (both total length and time) slightly better in all the cases, which is to be expected as there are not as many occlusions in the map as with "opaque" desks. We also see that exploration and planning together performs better in general and including the bias makes it more than 2.5 times better than the naive approach in terms of the path length.

In the 'Explore, then plan' case, the robot's visits to wastebins during the exploration do not count towards the mission satisfaction, in contrast to the 'simultaneous unbiased' case. This is one of the reasons the latter performs better, as expected. The 'simultaneous biased' version performs a lot better because it was able to visit a lot of wastebins (with the help of biasing) already during the exploration.

VI. CONCLUSIONS AND FUTURE WORK

We presented an online sampling-based algorithm capable of finding a trajectory in an *a priori* unknown environment that satisfies an scLTL specification. We enrich the RRG algorithm with functions that attempt to learn possible relations between labels of the environment and use such relations for biasing the search for a satisfying trajectory. The resulting paths

are significantly shorter than in straightforward sequential exploration followed by planning in a known space.

A few topics to be considered for future work include extending the approach to consider probabilistic relationships in the semantic abstraction of the system, as well as the extension to multi-agent systems.

REFERENCES

[1] AI Medina Ayala, Sean B Andersson, and Calin Belta. Temporal logic motion planning in unknown environments. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5279–5284. IEEE, 2013.

[2] Alper Aydemir, Andrzej Pronobis, Moritz Göbelbecker, and Patric Jensfelt. Active visual object search in unknown environments using uncertain semantics. *IEEE Transactions on Robotics*, 29(4):986–1002, 2013.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*, volume 26202649. 01 2008. ISBN 978-0-262-02649-9.

[4] Amit Bhatia, Lydia E Kavraki, and Moshe Y Vardi. Sampling-based motion planning with temporal goals. In *2010 IEEE International Conference on Robotics and Automation*, pages 2689–2696. IEEE, 2010.

[5] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding horizon "next-best-view" planner for 3D exploration. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1462–1468. IEEE, 2016.

[6] Yiannis Kantaros and Michael M Zavlanos. Sampling-based optimal control synthesis for multirobot systems under global temporal tasks. *IEEE Transactions on Automatic Control*, 64(5):1916–1931, 2018.

[7] Yiannis Kantaros and Michael M Zavlanos. STyLuS*: A Temporal Logic Optimal Control Synthesis Algorithm for Large-Scale Multi-Robot Systems. *The International Journal of Robotics Research*, 39(7):812–836, 2020.

[8] Yiannis Kantaros, Matthew Malencia, Vijay Kumar, and George J Pappas. Reactive Temporal Logic Planning for Multiple Robots in Unknown Environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11479–11485. IEEE, 2020.

[9] Sertac Karaman and Emilio Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 2222–2229. IEEE, 2009.

[10] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[11] Orna Kupferman and Moshe Y. Vardi. Model Checking of Safety Properties. In *CAV*, volume 1633 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 1999.

[12] Kim Guldstrand Larsen and Bent Thomsen. A Modal Process Logic. In *LICS*, pages 203–210. IEEE Computer Society, 1988.

[13] Xusheng Luo, Yiannis Kantaros, and Michael M Zavlanos. An Abstraction-Free Method for Multi-Robot Temporal Logic Optimal Control Synthesis. *arXiv preprint arXiv:1909.00526*, 2019.

[14] Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, page 4657, USA, 1977. IEEE Computer Society.

[15] Magnus Selin, Mattias Tiger, Daniel Duberg, Fredrik Heintz, and Patric Jensfelt. Efficient autonomous exploration planning of large-scale 3-D environments. *IEEE Robotics and Automation Letters*, 4(2):1699–1706, 2019.

[16] Pouria Tajvar, Fernando S Barbosa, and Jana Tumova. Safe Motion Planning for an Uncertain Non-Holonomic System with Temporal Logic Specification. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 349–354. IEEE, 2020.

[17] Cristian Ioan Vasile and Calin Belta. Sampling-based temporal logic path planning. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4817–4822. IEEE, 2013.

[18] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*, pages 146–151. IEEE, 1997.