2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Reactive Sampling-Based Temporal Logic Path Planning

Cristian Ioan Vasile and Calin Belta

***Abstract*— We develop a sampling-based motion planning algorithm that combines long-term temporal logic goals with short-term reactive requirements. The mission specification has two parts: (1) a global specification given as a Linear Temporal Logic (LTL) formula over a set of static service requests that occur at the regions of a known environment, and (2) a local specification that requires servicing a set of dynamic requests that can be sensed locally during the execution. Our method consists of two main ingredients: (a) an off-line sampling-based algorithm for the construction of a global transition system that contains a path satisfying the LTL formula, and (b) an on-line sampling-based algorithm to generate paths that service the local requests, while making sure that the satisfaction of the global specification is not affected. Building on our previous work [1], the focus of this paper is on the on-line part of the overall method.**

I. INTRODUCTION

A central problem in motion planning is to generate a path for a robot from an initial to a final desired position in an environment with obstacles. The most used algorithms are based on cell decompositions, potential fields, and navigation functions [2]. These methods, however, suffer from poor scalability with respect to the dimension of the configuration space. In order to overcome these limitations, probabilistically complete approaches based on randomized sampling, such as probabilistic roadmaps (PRM) [3] and rapidly exploring random trees (RRT) [4], and their asymptotically optimal counterparts, PRM* and RRT* [5], were proposed.

A recent trend in robot motion planning is the development of computational frameworks that allow for automatic deployment from rich, high-level, temporal logic specifications, e.g., "Visit A and then B or C infinitely often. Always avoid D. Never go to E unless F was reached before." It has been shown that temporal logics, such as Linear Temporal Logic (LTL), Computational Tree Logic (CTL), and μ-calculus, and their probabilistically versions (PLTL, PCTL), can be used as formal languages for motion planning [6], [7], [8], [9], [10]. Adapted model checking algorithms and automata game techniques [6], [11] were used to generate plans and control policies for finite models of robot motion. Such models were obtained through abstractions, which are essentially partitions of the robot configuration space that capture the ability of the robot to steer among the regions in the partition [12]. As a result, they suffer from the same scalability issues as the cell-based decomposition methods.

To generate motion plans and control strategies from rich task specifications for robots with large configuration spaces, a natural approach is to combine sampling-based motion planning with automata-based synthesis methods. The existing works in this area show that synthesis algorithms from specifications given in μ-calculus [9], [13] and LTL [1] can be adapted to scale incrementally with the graph constructed during the sampling process. However, these off-line algorithms assume that the robot moves in a static environment with a known, global map, and cannot react to events sensed locally during the deployment.

In this paper, we address the problem of generating a path for a robot required to satisfy a (global) LTL specification over some known, static service requests, while at the same time servicing a set of locally sensed requests ordered according to their priorities. We propose a random sampling approach that builds on our work from [1]. Our framework consists of two components: (1) an off-line algorithm that generates a finite transition system that contains a run satisfying the global specification, and (2) an on-line algorithm that finds local paths that satisfy both the local and the global specifications. The focus in this paper is on the on-line part of the problem, i.e., we assume that the finite transition system is available [1].

The main contribution of this work is a sampling-based, formal framework that combines infinite-time satisfaction of temporal logic global specifications with reactivity to requests sensed locally. Closely related works include [14], [15], [16], [17]. In [14], the authors consider global specifications given in the more restrictive scLTL fragment of LTL. To deal with the state-space explosion problem, they propose a layered path planning approach which uses a cell decomposition of the configuration space for high-level temporal planning and expansive space trees (EST) for kino-dynamic planning of the low-level, cell-to-cell motion. The on-line algorithm from [17] finds minimum violating paths for a robot when the global specification can not be enforced completely. In [15], [16], the global specifications are given in the GR(1) fragment of LTL, and on-line local

This work was partially supported by the ONR under grants MURI N00014-09-1051 and MURI N00014-10-10952 and by the NSF under grant NSF CNS-1035588.

The authors are with the Division of Systems Engineering, Boston University, Boston MA, {cvasile, cbelta}@bu.edu).

re-planning is done through patching invalidated paths based on μ-calculus specifications. Finally, the idea of using a potential function to enforce the satisfaction of an infinite-time specification through local decisions is inspired from [18].

II. PROBLEM FORMULATION

Consider a robot moving in an environment (workspace) $\mathcal{D}$ containing a set of disjoint regions of interest $\mathcal{R}_G$. We assume that the robot can precisely localize itself in the environment. There is a set of service requests Π_G at the regions in $\mathcal{R}_G$ and their location is given by a map $\mathcal{L}_G : \mathcal{R}_G \to 2^{\Pi_G}$. We assume that these regions as well as the labeling map are static and a priori known to the robot. We will refer to these as *global* regions and requests, because these are used to define the long-term goal of the robot's mission. An example of an environment with global regions and requests is shown in Fig. 1.

While the robot moves in the environment, it can locally sense a set of dynamic service requests denoted by Π_L and a particular type of avoidance request denoted by π_O, which captures moving obstacles, unsafe areas, etc. We assume $\Pi_G \cap (\Pi_L \cup \{\pi_O\}) = \emptyset$. A dynamic request from Π_L occurs at a point in the environment and has an associated *servicing radius*, which specifies the maximum distance from which the robot can service it. The servicing radius of a request is determined by its type (Π_L) and all servicing radii are known a priori. The robot may service a dynamic request by moving inside the request's servicing radius and performing an appropriate action. We assume that once a request is serviced, it disappears from the environment. The region around the robot in which the robot can sense a dynamic request, including π_O, is called the *sensing area* of the corresponding sensor. For simplicity, we assume that all sensors have the same sensing area. The sensing area may be of any shape and size provided that it is connected and full-dimensional (see Fig. 1). We assume that the avoidance request π_O is associated with whole regions, parts of which can be detected when they intersect with the robot's sensing area. For simplicity, we refer to regions satisfying π_O as *local obstacles*. The set of regions corresponding to local obstacles present in the environment at time $t \geq 0$ is denoted by $\mathcal{R}_L(t)$.

The mission specification is composed of two parts: a *global mission specification*, which is defined over the set of global properties Π_G, and a *local mission specification*, which specifies how on-line detected requests Π_L must be handled. The global mission specification, which defines the long-term motion of the robot, is given as an LTL$_{-X}$ (i.e., LTL without the "next" operator) formula Φ_G. [1]. When a robot passes over a global region, it is assumed that the robot services the requests associated with the region. Therefore, a path traveled by the robot generates a word over Π_G. A path is said to *satisfy the global mission specification* Φ_G if the corresponding word satisfies Φ_G. The local mission specification is given as a priority function $prio : \Pi_L \to \mathbb{N}$. We assume that $prio$ is an injective function that assigns lower values to higher priority requests.If the robot detects dynamic requests, it must go and service the request with the highest priority. If multiple requests have the same (highest) priority, then the robot can choose any one of them. Also, the robot must avoid all local obstacles marked by π_O.

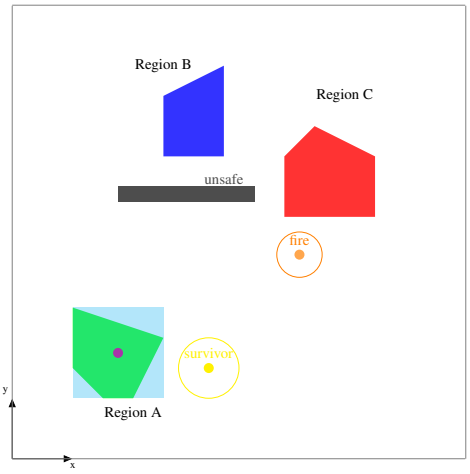


Fig. 1: Simplified representation of a disaster scenario considered in Example. 2.2. The environment contains three global regions A, B and C colored in green, blue and red, respectively. Three dynamic requests are also shown as colored points: a $survivor$ (yellow), a $fire$ (orange), and a local obstacle (black). The circles around them delimit the corresponding servicing areas. The initial position of the robot is shown in magenta and the cyan rectangle corresponds to its sensing area. In this figure the robot does not detect any dynamic request or local obstacles.

Let $\mathcal{C}$ be the compact configuration space of the robot and $\mathcal{H} : \mathcal{C} \to \mathcal{D}$ be a submersion that maps each configuration x to a position $y = \mathcal{H}(x) \in \mathcal{D}$. Formally, the problem can be formulated as follows:

Problem 2.1: Given a partially known environment described by $(\mathcal{D}, \mathcal{R}_G, \Pi_G, \mathcal{L}_G, \Pi_L)$, an initial configuration $x_0 \in \mathcal{C}$, an LTL$_{-X}$ formula Φ_G over the set of properties Π_G, and a priority function $prio : \Pi_L \to \mathbb{N}$, find an (infinite) path in the configuration space $\mathcal{C}$ originating at x_0 such that the path $y = \mathcal{H}(x)$ in the environment satisfies Φ_G and on-line detected dynamic

[1]Throughout the paper, we assume that the reader is familiar with LTL syntax and semantics and concepts from automata-based model checking, such as Buchi automata and product automata (see [19]).

requests, while avoiding local obstacles.

Example 2.2: Fig. 1 shows a simplified disaster response scenario, in which a point fully actuated robot is deployed in an environment where three global regions of interest A, B and C are defined. The set of dynamic requests is $\Pi_L = \{fire, survivor\}$ and the local obstacle is $\pi_O = unsafe$. If the robot detects requests $fire$ or $survivor$, it must service them by going within the corresponding servicing radii and initiating appropriate actions (i.e., extinguishing the fire and providing medical relief, respectively). If the robot detects the local obstacle $unsafe$ (shown in black in Fig. 1), the robot must avoid that region. The limited sensing area of the robot's sensors is depicted in Fig. 1 by a cyan rectangle.

The global mission specification is: *"Go to region A and then go to regions B or C infinitely often"*. This specification can be expressed in LTL$_{-X}$ as:

$$\Phi_G := \mathbf{GF}A \wedge \mathbf{G}(A\ \mathcal{U}\ (\neg A\ \mathcal{U}\ (B \vee C))) \qquad (1)$$

The local mission specification is to *"Extinguish fires and provide medical assistance to survivors, with priority given to survivors, while avoiding unsafe areas."*. Thus the priority function is defined such that $prio(survivor) = 0$ and $prio(fire) = 1$.

The proposed computational framework to solve Prob. 2.1 consists of two parts: (a) an off-line sampling-based algorithm to compute a global transition system $\mathcal{T}_G$ in the configuration space $\mathcal{C}$ of the robot that contains a path whose image in the workspace $\mathcal{D}$ satisfies the global mission specification Φ_G, and (b) an on-line sampling-based algorithm that computes at every time step a local control strategy that takes into account dynamic requests such that both local and global mission specifications are met. In our previous work [1], we developed an algorithm for the construction of $\mathcal{T}_G$. In short, this method builds on RRG [5] and provides a probabilistically complete algorithm that incrementally checks for the existence of satisfying paths when new samples are generated. In this paper, we only focus on the on-line part of the framework.

III. Problem solution

The approach taken in this paper is based on the RRT algorithm, a probabilistically complete sampling-based path planning method. We modify the standard RRT in order to find local paths which preserve the satisfaction of the global specification Φ_G, while servicing on-line requests and avoiding locally sensed obstacles.

To keep track of validity of samples (random configurations) with respect to the global specification Φ_G, we propose a method that combines the ideas presented in [20] on monitors for LTL formulae and [18] on potential functions. The problem considered in [20] is to decide as soon as possible if a given (infinite) word w satisfies a LTL formula ϕ. The main idea is to keep track of Büchi states corresponding to a finite prefix of w with respect to both ϕ and $\neg\phi$ concurrently. If one of the two sets of Büchi states corresponding to ϕ or $\neg\phi$ becomes empty, then we can conclude that the specification is either violated or satisfied. If both sets are non-empty then nothing can be said about $w \models \phi$. In our case, we just use half of a monitor, since we are interested only in checking if steering the robot to new samples violates Φ_G. The potentials functions approach described in [18] is used to address the problem of connecting the locally generated path to states in the global transition system such that Φ_G is satisfied.

In the following, we will denote by $\mathcal{B}$ the Büchi automaton encoding the LTL$_{-X}$ formula Φ_G. We use $\mathcal{P}_G = \mathcal{T}_G \times \mathcal{B}$ for the product automaton and $\mathcal{T}_L$ to denote the local transition system which is generated at each time step. An element of $\mathcal{D}$ will be called a *position*. The states of the $\mathcal{T}_G$ and $\mathcal{T}_L$ are configurations in $\mathcal{C}$. The weight of a transition of $\mathcal{T}_G$ or $\mathcal{T}_L$ is given by the distance between its endpoints in $\mathcal{C}$.

We make the following additional assumptions that are necessary in the technical treatment below. For a set $R \subseteq \mathcal{D}$ that is connected and has full dimension in $\mathcal{D}$, we assume that the inverse set $\mathcal{H}^{-1}(R)$ also has full dimension in $\mathcal{C}$. The global regions and local obstacles are connected sets with non-empty interior, (i.e. they have full dimension in $\mathcal{D}$). Also, all the connected regions in the free space, between global regions and obstacles, respectively, are full dimensional. This implies that all global regions, local obstacles, service areas for dynamic requests, and connected free space regions (all subsets of $\mathcal{D}$) have corresponding inverse sets (through $\mathcal{H}^{-1}$) of non-zero Lebesgue measure in $\mathcal{C}$. Note that these are just technical assumptions, which are normally made in sampling-based approaches, and we do not need to construct the inverse map $\mathcal{H}^{-1}$. In the sampling-based algorithms described below, we only need to check how the environment image of a configuration satisfies features of interest in the environment. Finally, we assume that the robot knows its configuration precisely and it can follow trajectories in the configuration space made of connected line segments. A path $\mathbf{x}$ in $\mathcal{C}$ is said to satisfy the specification Φ_G if the corresponding path $\mathbf{y} = \mathcal{H}(\mathbf{x})$ in $\mathcal{D}$ satisfies Φ_G. The initial configuration x_0 of the robot is known and $\mathcal{H}(x_0) = y_0$.

A. Potential functions

In [18] the authors define a potential function over the states of the product automaton between a transition system and a Büchi automaton. The potential function captures the distance from each state of the product to the closest final state. It can be thought of as a distance to satisfaction and resembles a Lyapunov function. We ex-

tend this notion to define potential functions on the states of the global transition system. This extension allows us to reason about the change of potential between nodes of $\mathcal{T}_G$ connected through local paths instead of a direct transition. The local paths are generated as branches of a tree by the proposed RRT-based algorithm. The definitions of self-reachable set and potential function for product automaton states are adapted from [18].

Let $\mathcal{P}_G = \mathcal{T}_G \times \mathcal{B} = (S_{\mathcal{P}_G}, S_{\mathcal{P}_{G0}}, \Delta_{\mathcal{P}_G}, \omega_{\mathcal{P}_G}, F_{\mathcal{P}_G})$ be a product automaton between a transition system $\mathcal{T}_G$ and Büchi automaton $\mathcal{B}$. We denote by $\mathcal{D}(p, p')$ the set of all finite trajectories between states $p, p' \in S_{\mathcal{P}_G}$, $\mathcal{D}(p, p') = \{p_1 \dots p_n | p_1 = p, p_n = p'; p_k \rightarrow_{\mathcal{P}_G} p_{k+1}, \forall k = 1, \dots, n-1; \forall n \geq 2\}$. A state $p \in S_{\mathcal{P}_G}$ is said to reach a state $p' \in S_{\mathcal{P}_G}$ if $\mathcal{D}(p, p') \neq \emptyset$. The length of a path is defined as the sum of the weights corresponding to the transitions it is composed of: $L(\mathbf{p}) = \sum_{k=1}^{n-1} \omega_{\mathcal{P}_G}(p_k, p_{k+1})$. For $p, p' \in S_{\mathcal{P}_G}$, the distance between p and p' is defined as follows:

$$d(p, p') = \begin{cases} \min_{\mathbf{p} \in \mathcal{D}(p,p')}(L(\mathbf{p})) & \text{if } \mathcal{D}(p, p') \neq \emptyset \\ \infty & \text{if } \mathcal{D}(p, p') = \emptyset \end{cases} \quad (2)$$

The weight function $\omega_{\mathcal{P}_G}$ is positive, because it is induced by the distance of the underlying (metric) space. This implies [18] that $d(p, p') > 0$ for all $p, p' \in S_{\mathcal{P}_G}$.

A set $A \subset S_{\mathcal{P}_G}$ is *self-reachable* if and only if all states in A can reach a state in A, i.e. for all $p \in A$ there is a state p' such that $\mathcal{D}(p, p') \neq \emptyset$.

Definition 3.1 (Potential function of states in $\mathcal{P}_G$): The potential function $V_{\mathcal{P}_G}(p)$, $p \in S_{\mathcal{P}_G}$ is defined as:

$$V_{\mathcal{P}_G}(p) = \begin{cases} \min_{p \in F^*_{\mathcal{P}_G}} d(p, p') & \text{if } p \in F^*_{\mathcal{P}_G} \\ 0 & \text{if } p \in F^*_{\mathcal{P}_G} \end{cases} \quad (3)$$

where $F^*_{\mathcal{P}_G} \subset F_{\mathcal{P}_G}$ is the maximal self-reachable set of final states of $\mathcal{P}_G$. The potential function is non-negative for all states of $\mathcal{P}_G$. It is zero for $p \in S_{\mathcal{P}_G}$ if and only if p is a final state and p can reach itself. Also, if $V_{\mathcal{P}_G}(p) = \infty$, then p does not reach any self-reachable final states.

Definition 3.2 (Potential function of states in $\mathcal{T}_G$): Let $x \in X$ and $B \subseteq \beta_{\mathcal{P}_G}(x)$. The potential function of x with respect to B is defined as: $V_{\mathcal{T}_G}(x, B) = min_{s \in B} V_{\mathcal{P}_G}((x, s))$ Also, the minimum potential of x is defined as $V^*_{\mathcal{T}_G}(x) = V_{\mathcal{T}_G}(x, \beta_{\mathcal{P}_G}(x))$. The minimum potential of a state x of $\mathcal{T}_G$ is the minimum potential of all states in $\mathcal{P}_G$ which correspond to x. The actual potential is defined to capture the fact that not all Büchi states may be available in order to achieve the minimum potential.

B. Satisfying local paths with respect to Φ_G

Local paths in our RRT based algorithm connect states of the global transition system. Let $x, x' \in \mathcal{T}_G$ and $\mathbf{x} = x_1 \dots x_n$ be a local path connecting $x_1 = x$ and $x_n = x'$ and $\mathbf{o} = o_1 \dots o_n$ be the output trajectory corresponding to $\mathbf{x}$ with respect to the global proprieties ($o_k \in 2^{\Pi_G}, \forall k = 1 \dots n$). We need to ensure that there is a satisfying run in $\mathcal{T}_G$ starting at x' after traversing $\mathbf{x}$. Thus, we need to consider two problems: (1) how to keep track of available Büchi states as local samples are generated and (2) how to connect a local path's endpoint (tree leaf) to the global transition system $\mathcal{T}_G$.

The first problem is solved by Alg. 1, which determines the set of Büchi states given a word w over 2^{Π_G}. Alg. 1 solves this problem by repeatedly computing the set of outgoing neighboring states of $\mathcal{B}$ for all states in the previous iteration. To check if a local path can be connected to the state $x_n = x' \in X$, we just need to verify that it has finite potential, i.e. $V_{\mathcal{T}_G}(x', B) < \infty$, where B is the set of available Büchi states after traversing $\mathbf{x}$, in this case $w = \mathbf{o}$. The second problem has a simple solution in this setting. We choose the state in $\mathcal{T}_G$ which has (finite) minimum potential after traversing a branch of the RRT tree. Also, the line segment between the leaf state from the tree and the state in $\mathcal{T}_G$ must be collision free (see Sec. III-C).

Algorithm 1: Tracking Büchi states of local samples

Input: $\mathcal{B}$ Büchi automaton for Φ_G, $w = \sigma_1 \dots \sigma_n$ finite word over 2^{Π_G}, B set of start Büchi states
Output: B_f set of final Büchi states after processing w

```
B_f ← B
for k ← 1 ... n do
    B' ← ∅
    foreach s ∈ B_f do
        B' ← B' ∪ {s' ∈ S_B | (s, s', σ_k) ∈ Δ_B}
    B_f ← B'
return B_f
```

C. On-line planning algorithm

The on-line planning algorithm, outlined in Alg. 2, is composed of an off-line preprocessing step of computing the potential function for $\mathcal{P}_G$ and the on-line loop. At each step of the loop, the robot scans for requests and local obstacles and checks if it needs to compute a new local path. Re-planning is performed in four cases: (1) if the current path is empty; (2) a higher priority request was detected; (3) the chosen request disappeared; and (4) the local path collides with a local obstacle. Büchi states are tracked starting from the initial configuration of the robot, corresponding to the initial state of $\mathcal{T}_G$. Map B is used to store the tracked Büchi states.

The local path planning algorithm is show in Alg. 3 and is based on RRT. The procedure incrementally constructs the local transition system $\mathcal{T}_L$. The initial (root) state of $\mathcal{T}_L$ is the current configuration of the robot x_c. The map $serv$ indicates whether a state or any of its ancestors serviced the on-line request with the highest

Algorithm 2: Planning algorithm

Input: $\mathcal{T}_G$ global transition system, $\mathcal{B}$ Büchi automaton for Φ_G, $\mathcal{P}_G = \mathcal{T}_G \times \mathcal{B}$, $prio$ priority function for on-line requests, x_0 initial configuration

1 Compute potential function $V_{\mathcal{P}_G}(\cdot)$
2 $path \leftarrow emptyList()$
3 $x_c \leftarrow x_0$; $B(x_c) \leftarrow \beta_{\mathcal{P}_G}(x_c)$
4 **while** *True* **do**
5 $\quad I \leftarrow getLocalRequests()$
6 $\quad$ **if** $checkPath(I, path) \vee \neg path.hasNext()$ **then**
7 $\quad\quad path \leftarrow planLocally(x_c, \mathcal{P}_G, \mathcal{B}, prior, I)$
8 $\quad x_n \leftarrow path.next()$
9 $\quad enforce(x_c \rightarrow x_n)$
10 $\quad x_c \leftarrow x_n$

Algorithm 3: Local path planning

Input: x_c current configuration of the robot, $\mathcal{P}_G = \mathcal{T}_G \times \mathcal{B}$, $\mathcal{B}$ Büchi automaton for Φ_G, $prior$ – on-line requests priority function, I sensed requests and local obstacles
Output: $path$ computed local control strategy

1 Construct $\mathcal{T}_L = (X_L, x_c, \Delta_L, \omega_L, \Pi_L \cup \{\pi_O\}, h_L)$ with x_c as initial state
2 $serv(x_c) \leftarrow \neg I.hasRequest()$
3 **while** $\nexists\, x_c \rightarrow^*_{\mathcal{T}_L} x_T \rightarrow x_G$ *w/* $V_{\mathcal{T}_G}(x_G, B(x_G)) < \infty \vee \neg serv(x_T)$ **do**
4 $\quad x_s \leftarrow generateSample(x_c, I.area)$
5 $\quad x_n \leftarrow nearest(\mathcal{T}_L, x_s)$
6 $\quad x \leftarrow steer(x_n, x_s, \eta)$
7 $\quad x \leftarrow label(x, I)$
8 $\quad B(x) \leftarrow trackBuchiStates(\mathcal{B}, h_L(x_n), B(x_n))$
9 $\quad$ **if** $B(x) \neq \emptyset \wedge isSimpleSegmnent(x_n, x) \wedge collisionFree(x_n, x)$ **then**
10 $\quad\quad serv(x) \leftarrow serv(x_n) \vee I.serviced(x, prior)$
11 $\quad\quad X_L \leftarrow X_L \cup \{x\}$; $\Delta_L \leftarrow \Delta_L \cup \{(x_n, x)\}$
12 **return** $x_c \rightarrow^*_{\mathcal{T}_L} x_T \rightarrow x_G$

priority. If there are no requests then $serv$ is true for all states of $\mathcal{T}_L$.

The construction of the RRT proceeds by generating a new random sample (line 4) inside the sensing area of the robot, steer the system towards it (lines 5–6) and checking if it is a valid state (lines 8–9). Samples are generated such that their images in $\mathcal{D}$ belong to the sensing area of the robot. The $nearest$ function (line 5) is a standard RRT primitive which returns the nearest state in $\mathcal{T}_L$ based on the distance function associated with $\mathcal{C}$. We assume that we have access to a $steer$ function (see Sec. II) which drives the system to a configuration $x \in \mathcal{C}$. x is the closest configuration to the new sample x_s which is at most η distance away from x_n, [4], [5]. The $label$ primitive function (line 7) is used to anotate x with the global properties it satisfies. x is valid if its corresponding set of Büchi states is non-empty and the line segment from its parent x_n to itself is a simple collision free line segment. Alg. 1 is used to compute the set of available Büchi states for x. The primitive function $isSimpleSegment$ is used to ensure that the set of global properties along the potential new transition (x_n, x) changes at most once. For more details, about this primitive see [1]. The $collisionFree$ primitive is used to check if the image in the workspace of the line segment $(x_n, x) \in \mathcal{C}$ collides with a local obstacle in $\mathcal{D}$. If these tests are passed, the procedure adds the state x and the transition (x_n, x) to $\mathcal{T}_L$ (line 11). Also the $serv$ map is updated by checking if either the parent state x_n (or some ancestor) or the state itself x has serviced the selected on-line request.

Also, we require that the state x_G of $\mathcal{T}_G$ have a lower (actual) potential than the last visited state x'_G of $\mathcal{T}_G$. This condition is not enforced, if the potential of x'_G is zero, but we still require $x_G \neq x'_G$.

Theorem 3.3: Let $\mathbf{x} = x_1, \ldots$ be an infinite path in $\mathcal{C}$ generated by Alg. 2 and $\mathbf{o} = o_1, \ldots$ be the corresponding (infinite) output word generated by traversing $\mathbf{x}$. If every call of Alg. 3 finishes in finite time, then $\mathbf{o}$ satisfies the global specification Φ_G, $\mathbf{o} \models \Phi_G$.

Remark 3.4: The complexity of the local path planning algorithm (Alg. 3) is the same as for the standard RRT. The functions $generateSample$, $steer$ and $nearest$ are stardard primitives [4], [5]. $label$, $isSimpleSegment$ and $collisionFree$ primitives and checking if an on-line request was serviced, can be reduced to collision detection in the lower dimensional workspace. Tracking Büchi states takes constant time ($O(1)$), because the global specification Φ_G is fixed.

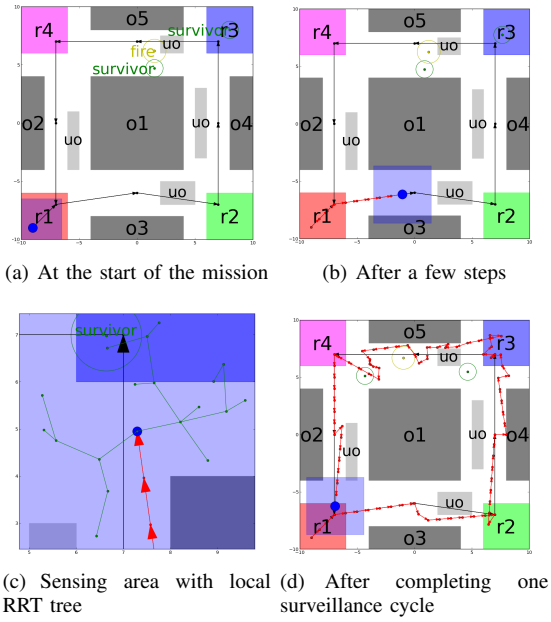
IV. CASE STUDY

The algorithms presented in this paper were implemented in Python2.7. The case study presented below was run on an Ubuntu 13.04 with 2GHz Intel Core2 Duo processor and 2GB of memory. For simplicity, we present a case study in which the robot is a fully actuated point in plane. Note that, since both the off-line and on-line algorithms are based on random sampling, the method can be used for systems with large configuration spaces. In fact, in [1], where we presented the off-line part of the method, we included a case study in a 20-dimensional configuration space.

Here we consider the configuration space depicted in Fig. 2(a). The initial configuration is $x_0 = (-9; -9)$. The global specification is to visit regions $r1$, $r2$, $r3$ and $r4$ infinitely many times while avoiding regions $o1$, $o2$, $o3$, $o4$ and $o5$. The corresponding LTL$_{-X}$ formula is $\Phi_G = \mathbf{G}(\mathbf{F}r1 \wedge \mathbf{F}r2 \wedge \mathbf{F}r3 \wedge \mathbf{F}r4 \wedge \neg(o1 \vee o2 \vee o3 \vee o4 \vee o5))$. There are four local obstacles labeled uo and three dynamic requests: two $survivor$ requests

and a $fire$ request. The three dynamic requests have a cyclic motion at a lower speed than that of the robot. The maximum distance traveled by the robot in one discrete time step is $\eta = 1$ (see the steer primitive in Alg. 3, line 6). The priority function $prior$ is defined such that $survivor$ request have higher priority than $fire$ request.

A solution to this problem is shown in Fig. 2. The product automaton has a single accepting state, which corresponds to $x_{accept} = (-7, 0)$ in $\mathcal{T}_G$. The robot must visit x_{accept} infinitely many times. In each surveillance cycle, the three dynamic requests described above are created. We ran the path planning algorithm in order to complete 100 surveillance cycles. During the simulation, the local path planning algorithm (Alg. 3) was executed 5947 times. The overall execution time dedicated to local planning (lines 7–8 of Alg. 2) for a single surveillance cycle was on average 0.743 seconds (std. 0.216). The mean size of the generated local transition system $\mathcal{T}_L$ was 7.6 (std. 13.15). The path planning algorithm computed local paths which serviced 292 on-line requests from a total of 296 detected.



(a) At the start of the mission (b) After a few steps

(c) Sensing area with local RRT tree (d) After completing one surveillance cycle

Fig. 2: The environment contains four global regions of interest $r1$ (red), $r2$ (green), $r3$ (blue) and $r4$ (magenta), five global obstacles $o1, \ldots, o5$ (dark grey) and four local a priori unknown obstacles labeled uo (light grey). There are also three dynamic requests, two $survivor$ (green) and a $fire$ (yellow). The circles around the on-line requests delimit their corresponding service area. The sensing range of the robot is shown as a light blue rectangle (length of its side is 5) around the current position of the robot (blue dot), Fig. 2(b). The black arrows and dots represent the global transition system $\mathcal{T}_G$. The trajectory of the robot is shown as a sequence of red arrows. Fig. 2(d) shows the trajectory of the robot after completing a surveillance cycle. Fig. 2(c) is a close up view of the sensing area of the robot at position $(4.9, 7.3)$ where an RRT tree is generated. The red arrows mark the trajectory of the robot, and the black ones belong to $\mathcal{T}_G$.

REFERENCES

[1] C. Vasile and C. Belta, "Sampling-Based Temporal Logic Path Planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[2] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Boston: MIT Press, 2005.

[3] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[4] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *IEEE International Conference on Robotics and Automation*, 1999, pp. 473–479.

[5] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, June 2011.

[6] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *IEEE International Conference on Robotics and Automation*, 2007.

[7] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding Horizon Temporal Logic Planning for Dynamical Systems," in *Conference on Decision and Control*, 2009, pp. 5997 –6004.

[8] A. Bhatia, L. Kavraki, and M. Vardi, "Sampling-based motion planning with temporal goals," in *IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 2689–2696.

[9] S. Karaman and E. Frazzoli, "Sampling-based Motion Planning with Deterministic μ-Calculus Specifications," in *IEEE Conference on Decision and Control*, Shanghai, China, 2009.

[10] X. C. Ding, M. Kloetzer, Y. Chen, and C. Belta, "Formal Methods for Automatic Deployment of Robotic Teams," *IEEE Robotics and Automation Magazine*, vol. 18, pp. 75–86, 2011.

[11] Y. Chen, J. Tumova, and C. Belta, "LTL Robot Motion Control based on Automata Learning of Environmental Dynamics," in *IEEE International Conference on Robotics and Automation*, Saint Paul, MN, USA, 2012.

[12] 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.

[13] S. Karaman and E. Frazzoli, "Sampling-based Optimal Motion Planning with Deterministic μ-Calculus Specifications," in *American Control Conference*, 2012.

[14] M. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative Temporal Motion Planning for Hybrid Systems in Partially Unknown Environments," in *ACM International Conference on Hybrid Systems: Computation and Control*, ACM. Philadelphia, PA, USA: ACM, March 2013, pp. 353–362.

[15] S. C. Livingston and R. M. Murray, "Just-in-time synthesis for motion planning with temporal logic," in *International Conference on Robotics and Automation*, 2013.

[16] S. C. Livingston, P. Prabhakar, A. B. Jose, and R. M. Murray, "Patching task-level robot controllers based on a local μ-calculus formula," in *International Conference on Robotics and Automation*, 2013.

[17] J. Tumova, L. Reyes-Castro, S. Karaman, E. Frazzoli, and D. Rus, "Minimum-violating planning with conflicting specifications," in *American Control Conference*, 2013.

[18] X. C. Ding, M. Lazar, and C. Belta, "Receding Horizon Temporal Logic Control for Finite Deterministic Systems," in *American Control Conference*, Montreal, Canada, 2012.

[19] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT Press, 2008.

[20] A. Bauer, M. Leucker, and C. Schallhart, "Runtime Verification for LTL and TLTL," Institut für Informatik, Technische Universität München, Tech. Rep. TUM-I0724, December 2007.