Reactive Sampling-Based Temporal Logic Path Planning

Cristian Ioan Vasile and Calin Belta

{cvasile, cbelta }@bu.edu

Division of Systems Engineering, Boston University, Boston MA, USA

Abstract

We developed 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. The proposed computational framework 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. The off-line algorithm has three main features. First, it is incremental, in the sense that the procedure for finding a satisfying path at each iteration scales only with the number of new samples generated at that iteration. Second, the underlying graph is sparse, which guarantees the low complexity of the overall method. Third, it is probabilistically complete. We also provide a conditional result showing that the incremental checking procedure has the best possible complexity bound. The on-line algorithm leverages ideas of potential functions, which ensure progress towards satisfaction of the global specification, and on monitors for LTL. Examples illustrating the usefulness and the performance of the framework are included.

Setup

An **environment** is described by a tuple $(\mathcal{D}, \mathcal{R}_G, \Pi_G, \mathcal{L}_G, \Pi_L \cup \{\pi_O\})$, where:

- $\mathcal{D}$ is the workspace – assume that the robot can precisely localize itself in the environment
- $\mathcal{R}_G$ – set of disjoint regions of interest in $\mathcal{D}$
- Π_G – set of service requests at the regions in $\mathcal{R}_G$
- $\mathcal{L}_G : \mathcal{R}_G \to 2^{\Pi_G}$ – labeling map giving the location of the requests
- assume regions $\mathcal{R}_G$ and labeling map $\mathcal{L}_G$ are static and a priori known
- Π_L – set of dynamic service requests which are locally sense
- π_O – special request: moving obstacles, unsafe areas, etc.
- each dynamic request from Π_L has an associated *servicing radius* – once serviced, they disappears from the environment
- local requests: detected only inside the *sensing area* of the robot
- local obstacles: only the part inside the sensing area is detected

The **mission specification** is composed of two parts:

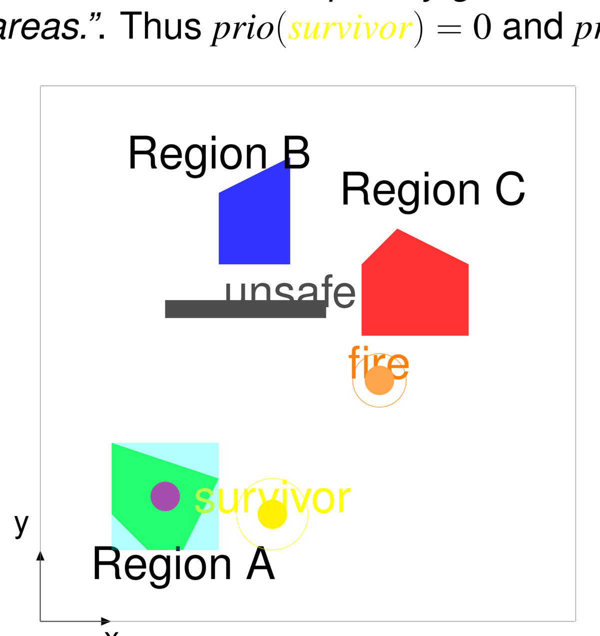
1. The **global mission specification** defines the long-term motion of the robot and is given as an LTL$_{-X}$ formula Φ_G. A path of the robot satisfies Φ_G if the (infinite) sequence of requests associated with the global regions the path passes through, satisfies Φ_G.
2. The **local mission specification** specifies how on-line detected requests Π_L must be handled and is given as a priority function $prio : \Pi_L \to \mathbb{N}$. Lower values indicate higher priority requests. The robot must go and service the dynamic request with the highest priority, if at least one is detected, while avoiding all local obstacles marked by π_O. Priority tiebreaking is random.

Planning is performed in the configuration space $\mathcal{C}$ of the robot and we use a submersion $\mathcal{H} : \mathcal{C} \to \mathcal{D}$ to map each configuration x to a position $y = \mathcal{H}(x)$.

Example Scenario

Consider a simplified disaster response scenario, in which a fully actuated point robot is deployed:

- A, B and C are the global regions;
- The set of dynamic requests is $\Pi_L = \{fire, survivor\}$ and the local obstacle is $\pi_O = unsafe$. The circles around the dynamic requests are the servicing radii and the limited sensing area is depicted by a cyan rectangle.
- The global mission specification is: *"Go to region A and then go to regions B or C infinitely often"*, which can be expressed as: $\Phi_G := \mathbf{GF}A \wedge \mathbf{G}(A \Rightarrow (\neg A\ \mathcal{U}\ (B \vee C)))$
- The local mission specification is to *"Extinguish fires and provide medical assistance to survivors, with priority given to survivors, while avoiding unsafe areas."*. Thus $prio(survivor) = 0$ and $prio(fire) = 1$.



Problem (Reactive Temporal Logic Path Planning Problem)

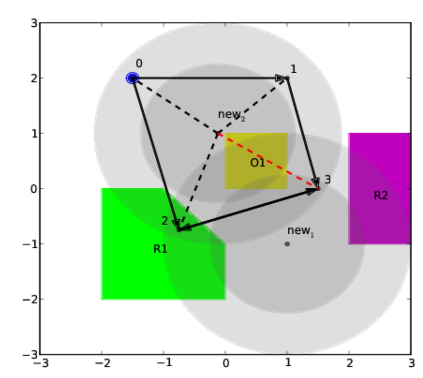
Given a partially known environment described by $(\mathcal{D}, \mathcal{R}_G, \Pi_G, \mathcal{L}_G, \Pi_L)$, an initial configuration $x_0 \in \mathcal{C}$, an LTL$_{-X}$ formula Φ_G over the set of properties Π_G, and a priority function $prio : \Pi_L \to \mathbb{N}$, find an (infinite) path in the 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 requests, while avoiding local obstacles.

Off-line Algorithm

The off-line algorithm is used to generate the global transition system $\mathcal{T}_G$. The procedure is a modified version of RRG with the following properties:

1. **Sparsity**: The generated transition system $\mathcal{T}_G$ is "sparse" (metric), i.e. the minimum distance between any two states of $\mathcal{T}$ is greater than a prescribed function dependent only on the size of $\mathcal{T}_G$ $(\min_{x,x' \in \mathcal{T}_G} \|x - x'\|_2 \geq \eta(|\mathcal{T}_G|))$. The metric sparsity implies that $\mathcal{T}_G$ is a sparse graph. Sparsity is desired because $\mathcal{T}_G$ is used in the on-line part of the framework. The environment is partially know by the robot before the start of the mission. Since transitions of $\mathcal{T}_G$ may need to be locally re-planned on-line, $\mathcal{T}_G$ must only capture the essential features of $\mathcal{D}$ such that Φ_G is satisfied. Sparsity also plays an important role in establishing the complexity bounds for the incremental search algorithm for a satisfying path.

   In order to obtain a sparse $\mathcal{T}_G$, we define a new primitive $far(x, \eta_1, \eta_2)$. The **far function** $far : \mathcal{C} \times \mathbb{R} \times \mathbb{R} \to 2^X$ returns the set of states from $\mathcal{T}_G$ that are at most at η_2 distance away from x. However, it returns an empty set if any state of $\mathcal{T}_G$ is closer to x than η_1. The **bound functions** must satisfy $\eta_1(k) < \eta_2(k)$ and also $c\,\eta_1(k) > \eta_2(k)$, for some finite $c > 1$ and all $k \geq 0$. Also, η_1 tends to 0 as k tends to infinity.

2. **Incremental**: The RRG-based algorithm for generating $\mathcal{T}_G$ is complemented with an incremental search algorithm for satisfying paths. The algorithm is based on the incremental update of the product automaton $\mathcal{P}_G = \mathcal{T}_G \times \mathcal{B}$, where $\mathcal{B}$ is the Büchi automaton encoding Φ_G, and incremental maintenance of the *strongly connected components* of $\mathcal{P}_G$ using the algorithm from [4]. We show that the overall execution time of the incremental search algorithm is $O(n^{\frac{3}{2}})$, where $n = |\mathcal{T}_G|$ is the number of states added during the search. We also show that this is the the best possible complexity for sparse transition systems among incremental algorithms, which have a "locality" property [4]. The maximum number of neighbors for a state of $\mathcal{T}_G$ is of order 2^d, where d is the dimension of $\mathcal{C}$, but the actual number is usually much lower.
3. **Probabilistically Complete**: The proposed algorithm is probabilistically complete, i.e. the probability that a satisfying path is found approaches 1 as the number of samples increases.

On-line Algorithm

The on-line planning algorithm is based on RRT, which we modify 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 [5] on monitors for LTL formulae and [3] on potential functions. Monitors are used to decide whether an infinite word satisfies or violates an LTL formula based on a finite prefix of it. 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 [3] 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.

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. The potential function $V_{\mathcal{P}_G}(p)$, $p \in S_{\mathcal{P}_G}$ is defined as the minimum (graph) distance between p and a final state in $F^*_{\mathcal{P}_G}$, where 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 similar to a Lyapunov function and is non-negative for $\mathcal{P}_G$. It is zero for $p \in S_{\mathcal{P}_G}$ if and only if p is a final state and p can reach itself or a self-reachable final state. Also, if $V_{\mathcal{P}_G}(p) = \infty$, $p \in S_{\mathcal{P}_G}$, then p does not reach any self-reachable final states. The potential function takes $O(|S_{\mathcal{P}_G}| \log |S_{\mathcal{P}_G}| + |\Delta_{\mathcal{P}_G}|)$ to compute. We extend this definition to the states of $\mathcal{T}_G$. Thus, the potential of a state x w.r.t. a set of Büchi states B is the minimum potential of all the states in $\mathcal{P}_G$ formed from x and B. The potential $V_{\mathcal{T}_G}(x, B)$ is defined to capture the fact that not all Büchi states may be available.

Planning Algorithm

Given the global LTL$_{-X}$ specification Φ_G, the priority function for on-line requests $prio$ and the initial configuration of the robot x_0, the steps of the path planning procedure are:

1. Convert Φ_G to Büchi automaton $\mathcal{B}$
2. Compute $\mathcal{T}_G$ and $\mathcal{P}_G = \mathcal{T}_G \times \mathcal{B}$ starting at x_0 using the RRG
3. Compute potential function $V_{\mathcal{P}_G}(\cdot)$
4. $path \leftarrow emptyList()$; $x_c \leftarrow x_0$; $B(x_c) \leftarrow \beta_{\mathcal{P}_G}(x_c)$
5. **Repeat indefinitely**:
   - 5.1 $I \leftarrow getLocalRequests()$
   - 5.2 **If** $(checkPath(I, path) \vee \neg path.hasNext())$ **then** $path \leftarrow planLocally(x_c, \mathcal{P}_G, \mathcal{B}, prior, I)$;
   - 5.3 $x_n \leftarrow path.next()$; $enforce(x_c \to x_n)$; $x_c \leftarrow x_n$;

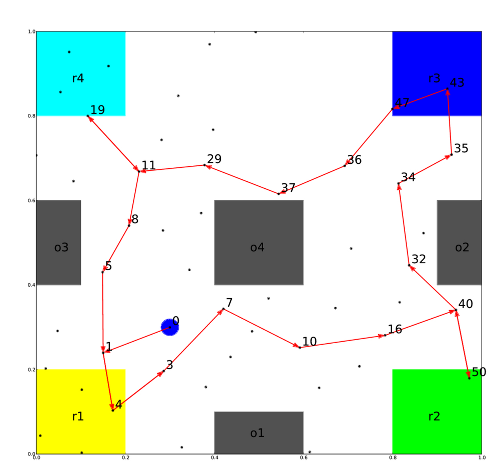
In the above procedure, local paths are generated by the RRT-based procedure $planLocally()$ such that they satisfy Φ_G. This requirement is achieved by tracking of Büchi states for local samples and connecting the leafs of the local tree to states in $\mathcal{T}_G$ which have (finite) minimum potential after traversing the corresponding branch of the local tree. Also, the line segment between the leaf state from the tree and the state in $\mathcal{T}_G$ must be collision free w.r.t. local obstacles.

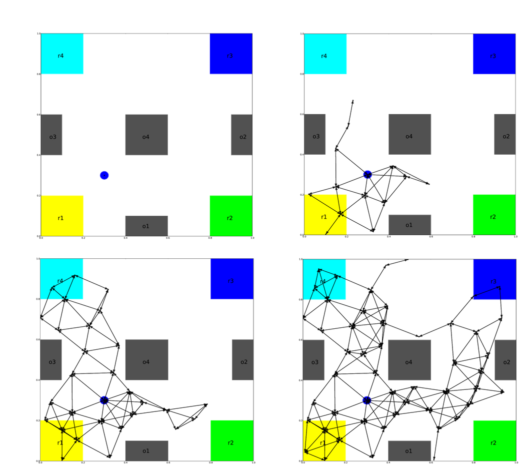
Theorem: Using the above planning procedure, the returned infinite path $\mathbf{x} = x_1, \ldots$ in $\mathcal{C}$ satisfies the global mission specification Φ_G if every call of the local planner $planLocally()$ terminates in finite time.

Case Studies

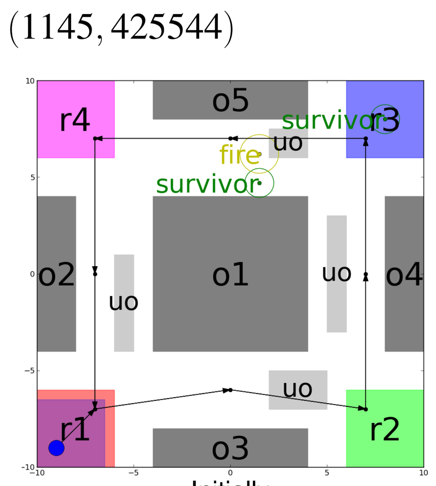
Case Study 1 (Off-line algorithm):

- 2D configuration space w/ $x_0 = (0.3; 0.3)$
- specification: "visit regions $r1$, $r2$, $r3$ and $r4$ infinitely many times while avoiding regions $o1$, $o2$, $o3$ and $o4$"
- LTL_{-X} $\phi_1 = \mathbf{G}(\mathbf{F}r1 \wedge (\mathbf{F}r2 \wedge (\mathbf{F}r3 \wedge (\mathbf{F}r4))) \wedge \neg(o1 \vee o2 \vee o3 \vee o4))$
- mean execution time: 6.954 sec; mean $|\mathcal{T}_G| = (51, 277)$; mean $|\mathcal{P}_G| = (643, 7414)$; $|\mathcal{B}| = (20, 155)$, where $\mathcal{B}$ encoding ϕ_1.

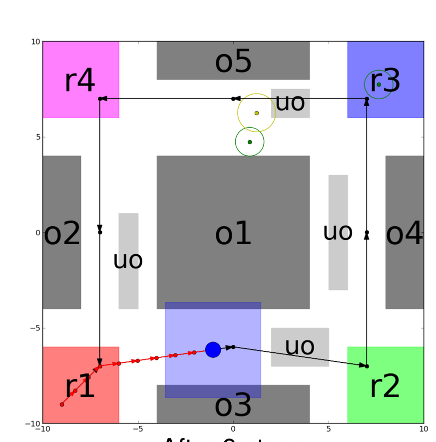




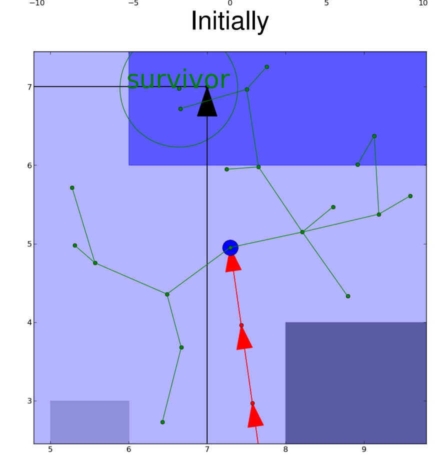
- 10D: mean time 16.75 sec, mean $|\mathcal{T}_G| = (69, 1578)$ and mean $|\mathcal{P}_G| = (439, 21300)$
- 20D: mean time 7.45 minutes, mean $|\mathcal{T}_G| = (414, 75584)$ and mean $|\mathcal{P}_G| = (1145, 425544)$



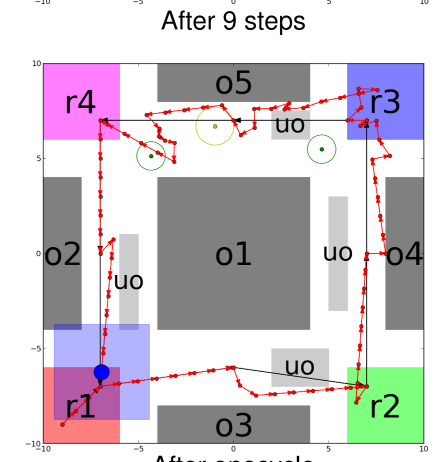
Initially



After 9 steps



Local RRT tree



After onecycle

Case Study 2 (On-line algorithm):

- four local obstacles (uo)
- three dynamic requests: two $survivor$ and a $fire$
- $survivor$ requests have higher priority than $fire$ requests
- $planLocally()$ was executed 5947 times, took 0.743 sec on average (std. 0.216, min. 0.436sec, max. 1.645sec)
- mean $|\mathcal{T}_L| = 7.6$ (std. 13.15, max. 165)
- serviced 292 on-line requests from a total of 296 detected

Acknowledgment

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.

References

- Cristian Vasile and Calin Belta. Sampling-Based Temporal Logic Path Planning. IROS, Tokyo, 2013.
- Cristian Vasile and Calin Belta. Reactive Sampling-Based Temporal Logic Path Planning. ICRA, Hong Kong, 2014.
- Xu Chu Ding, Mircea Lazar, and Calin Belta. Receding Horizon Temporal Logic Control for Finite Deterministic Systems, ACC, 2012.
- B. Haeupler, et. al. Incremental Cycle Detection, Topological Ordering, and Strong Component Maintenance. ACM Trans. Alg., 2012.
- Andreas Bauer, et. al. Runtime Verification for LTL and TLTL. TUM-I0724, Institut für Informatik, Tech. Universität München, 2007.