Planning Under Topological Constraints Using Beam-Graphs

Venkatraman Narayanan[†], Paul Vernaza[†], Maxim Likhachev[†], and Steven M. LaValle[‡]

***Abstract*— We present a framework based on graph search for navigation in the plane with a variety of topological constraints. The method is based on modifying a standard graph-based navigation approach to keep an additional state variable that encodes topological information about the path. The topological information is represented by a sequence of *virtual sensor beam* crossings. By considering classes of beam crossing sequences to be equivalent under certain equivalence relations, we obtain a general method for planning with topological constraints that subsumes existing approaches while admitting more favorable representational characteristics. We provide experimental results that validate the approach and show how the planner can be used to find loop paths for autonomous surveillance problems, simultaneously satisfying minimum-cost objectives and in dynamic environments. As an additional application, we demonstrate the use of our planner on the PR2 robot for automated building of 3D object models.**

I. Introduction

We consider in this work the problem of planning paths in the plane subject to different kinds of *topological constraints*. Fig. 1 demonstrates an application of planning with topological constraints to the problem of planning for Unmanned Aerial Vehicle (UAV) surveillance. In this scenario, we wish to find a path that takes the UAV on a circuit beginning and ending at its home base, subject to the constraint that it must observe some set of regions of interest (ROI). We can achieve this by planning with a constraint imposed on the way the path winds around the ROI.

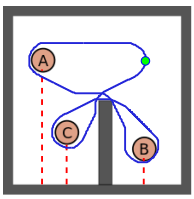
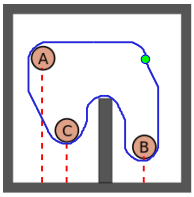
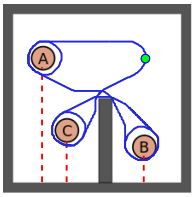
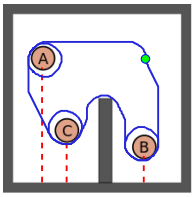
Search-based planning with winding constraints in particular has been considered in previous work [1, 9, 5, 13]. In this work, we propose a generalization of these methods that allows for planning with winding constraints as well as various other kinds of topological constraints in a search-based framework. Our approach additionally features a simplified state representation that captures the bare minimum information necessary to plan with these kinds of constraints, which enables more efficient implementation without any numerical stability issues such as can arise in other methods [1, 13]. The approach is also quite intuitive. Inspired by similar ideas in computational geometry [3, 4, 2] and combinatorial filtering [12], we imagine that *virtual sensor beams* emanate from obstacles in our environment. Our approach then consists of simply applying a graph-based planner with an augmented state in which we record sequences of beam crossing events. By declaring certain kinds of states to be equivalent via different relations on these sequences, we can enforce various sorts of topological constraints on the generated path. Table I demonstrates some of the different kinds of constraints that can be enforced in this way (see Sec. IV-C for details).

[†]V. Narayanan, P. Vernaza and M. Likhachev are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, USA venkatraman at cmu.edu, {pvernaza,maxim} at cs.cmu.edu

[‡]S.M. LaValle is with the Department of Computer Science, University of Illinois, Urbana, USA lavalle at uiuc.edu

TABLE I

Examples of different topological constraints that can be handled by planning using beam-graphs

Constraint in words (algebraic topology equivalent, if any)	String relations applied	Illustrative figure
Find a loop that crosses beams A, B and C, in the specified order (Homotopy)	$xx' \sim x'x \sim \varepsilon$	
Find a loop that crosses beams A, B and C in some order (Homology)	$xx' \sim x'x \sim \varepsilon$ $yx \sim xy$	
Obtain 360° views of regions A, B and C, in the specified order (Homotopy)	$xx' \sim x'x \sim \varepsilon$	
Obtain 360° views of regions A, B and C, in some order	$xx' \sim x'x \sim \varepsilon$, $yyxx \sim xxyy$ (only at goal location)	

II. Method Overview

We illustrate the idea of planning with virtual sensor beams by a simple example, illustrated in Fig. 2. Shown in the figure are the virtual sensor beams (in red dotted lines) and three paths, that represent three of the many possible paths that will bring the robot from its start position to a goal position. Each path may be associated with a string that records the sequence and direction in which it crossed each of the beams. For path 1, this string would correspond to ab, while path 3's string would be cc, and path 2's string

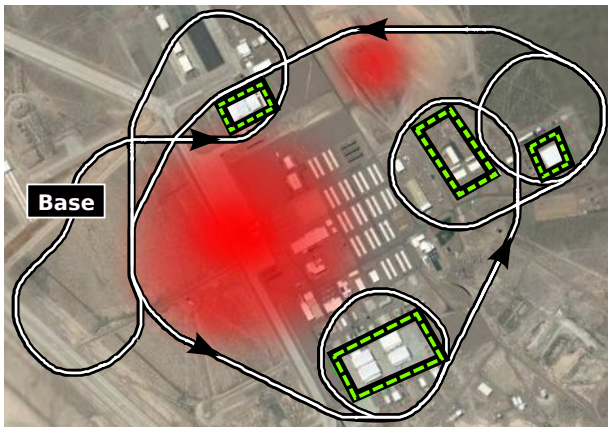


Fig. 1. An example reconnaissance mission where a UAV has to loop around certain regions of interest (marked by green dashed lines) and return back to its base. The red zones indicate radar installations, whose strength falls off with distance, that the UAV must try to avoid.

might be represented as $aa'ab$, where a' indicates that a was traversed in the direction contrary to the arrow.

Path 1 and path 3 have clearly different topological properties, and this is reflected in the fact that their associated strings are different. Paths 1 and 2 are topologically similar in the sense that one can be continuously deformed to another, but this similarity is not reflected in equivalence of their associated strings. If we choose, however, we can easily rectify this by considering any string of the form xx' or $x'x$ to be equivalent to the empty string. In this case, $aa'ab \sim ab$, where $\sim$ denotes equivalence. In fact, as we will discuss later, if we apply this equivalence relation, one path can be continuously deformed to another *iff.* their associated strings are equivalent. Thus, we can search for paths up to this notion of equivalence by searching for paths with certain strings of beam crossings. Furthermore, different topological constraints can be obtained by applying different notions of equivalence of strings.

Fig. 3 illustrates how different behaviors can be obtained by applying different sets of equivalence relations. The top and bottom images show the paths returned by the planner for *homotopy* and *homology* [7] constraints, respectively, when planning for the same goal string $abccdd$. One can clearly see that the homotopy constraint is stricter, since the path crosses the beams exactly in the order as specified in the goal string. The homology constraint, on the other hand is a weaker one, and the planner now searches over all possible permutations of the goal string and picks the one that yields the least cost solution. If we identify each ROI with the beam that emanates from it, the homology constraint $abccdd$ can be interpreted as "wind counter-clockwise around regions a and b once, and regions c and d twice, in some order that minimizes the length of the path".

Depending on the problem being solved, homotopy constraints might be more useful than homology constraints, or vice versa. One example where the homotopy constraint is more useful is the UAV reconnaissance problem, where the

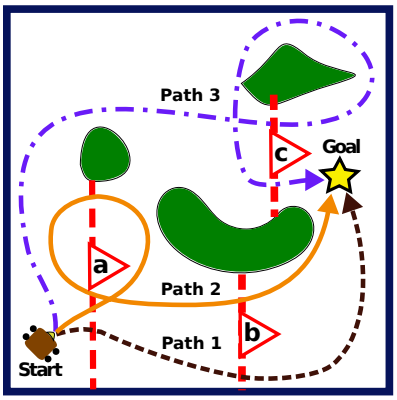
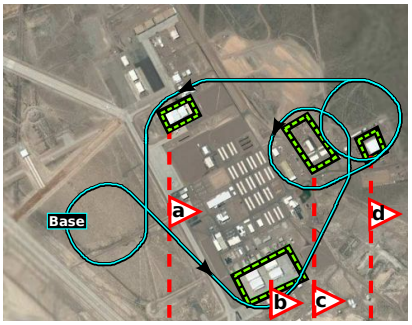
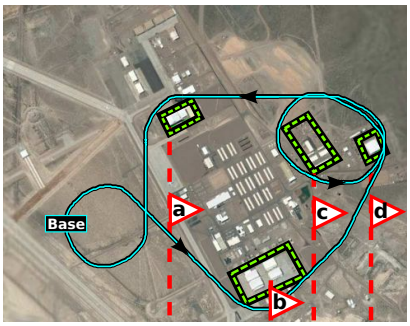


Fig. 2. Illustration of how sequences of beam crossings can capture topological information



(a)



(b)

Fig. 3. Effects of manipulating the string of beam crossings: (a) Planning for the goal string $abccdd$ with relations corresponding to homotopy constraints. (b) Planning for the same goal string $abccdd$, but with relations corresponding to homology constraints

UAV is required to obtain unobstructed 360° views of the ROI. More generally, planning to obtain 360° views of the ROI can be interpreted as a visibility constraint that requires the robot to obtain unobstructed views of every point on the periphery of an ROI. This can be achieved simply by planning for a sequence of beam crossings corresponding to the ROI, except that every beam crossing is double counted. For instance, in the UAV reconnaissance example, if our task were to obtain unobstructed views of every point on the boundary of regions a, b, c and d, we would plan with homotopy constraints for the goal string $aabbccdd$, as shown in Fig. 1.

III. Related Work

As mentioned in the introduction, our work is related to existing search-based methods for planning with winding constraints [1, 9, 5, 13]. However, we achieve planning for a strictly more general class of topological constraints. Furthermore, we note that some of these previous methods are imprecise in that they conflate the notions of *homotopy* (in which winding order matters) and *homology* (in which order does not matter) [7]; the methods [1, 9, 5, 13] can only enforce homology constraints, which are not as useful for certain types of problems. We will see that our method can enforce either type of constraint (as well as many others) simply by applying different equivalence relations on the strings of beam crossings.

Our method is inspired by the work on combinatorial filters by Tovar et al. [12], in which a sensor beam model is used to reconstruct paths of moving bodies, up to various topological resolutions, just using information from the beam crossings. This work makes explicit the relationship between sequences of beam crossings and homotopy/homology classes. We will revisit this topic in Sec. IV-A. A similar ray construction method appears in the context of planning in [8], but that work solves a simple planar navigation problem without topological constraints; homotopy information is used in an attempt to accelerate planning for this problem.

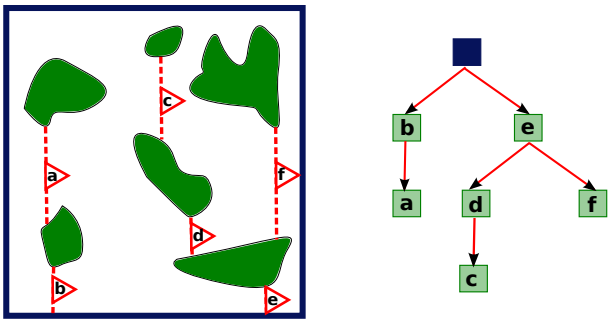
Finding shortest paths subject to homotopy constraints is a well-known topic in computational geometry, and efficient algorithms exist to solve special cases of the problem [3, 4]. The search-based method presented here is more general in that it handles a wide variety of constraints; additionally, our method is better suited to planning for robotics applications, as it can easily handle additional complications such as path curvature constraints and cost functions that vary arbitrarily in space and time.

IV. Technical Details

We now discuss the technical details associated with the construction of virtual beams and the beam-graph.

A. Virtual Beams Construction

A key component of our method is the construction of an appropriate set of virtual sensor beams. Although these may be constructed in an arbitrary fashion, constructing them in a particular way will allow us to later relate sequences of beam crossings to the aforementioned topological notions of homology and homotopy classes. This construction, illustrated in Fig. 4(a), consists simply of drawing a vertical line from the lowest point on each ROI (ties broken arbitrarily) until it intersects another ROI or the boundary of the environment. The set of lines so obtained constitutes the set of virtual sensor beams. Each beam is also associated with a normal direction—hereby assumed to be the direction pointing directly to the right—and a unique character c. Moving across the beam with character c in the normal direction is associated with the *beam crossing string* c, while moving across the beam in the opposite direction is associated with the string c'.



(a) (b)

Fig. 4. Construction of virtual beams: (a) The regions in green represent the ROI, and the red dotted lines are the virtual beams corresponding to those ROI. (b) The beam tree corresponding to the virtual beams construction in (a)

In order to establish the aforementioned topological connections, it will be useful to regard the set of virtual beams as a spanning tree on the set of nodes consisting of ROI and the boundary, as illustrated in Fig. 4(b). The spanning tree property is demonstrated in the following theorem.

Theorem 1. *Assume we have a set of ROI $\mathcal{R}$ in general position. Then the previous construction yields a spanning tree on $\mathcal{R}$ and the boundary, where the boundary and the elements of $\mathcal{R}$ are considered nodes, and beams are considered edges of a graph.*

Proof. Let $y(p)$ denote the vertical position of point p. We declare that ROI A is *lower* than ROI B iff. $\exists p \in A \mid y(p) < y(q), \forall q \in B$. This defines a total order on $\mathcal{R}$ by the assumption of general position. We then proceed by induction, assuming that at step k of the construction, we have constructed a spanning tree on the k lowest ROI (plus the boundary). This is trivially true for $k = 0$, as the graph consists of a single node (the boundary). Assuming the hypothesis is true at step k, we then drop a beam from the lowest point on the $k+1$-lowest ROI. This beam can only intersect a lower ROI, which must be in the spanning tree on the k lowest ROI by the inductive hypothesis. The resulting graph therefore consists of a single leaf added to the existing spanning tree, yielding a spanning tree on the $k+1$-lowest ROI. ■

B. Beam-Graph Construction

Each state s in the graph is a combination of $x(s)$, the geometric configuration of the robot in the environment; and $b(s)$, a string that keeps track of the topological information. As a common example, $x(s)$ could be a vector containing the position and heading of a robot on the plane. A state $\hat{s}$ is declared a successor of state s iff. there exists a motion m taking the robot from $x(s)$ to $x(\hat{s})$, and $b(\hat{s}) = b(s) + w(s, m)$, where $w(s, m)$ is the ordered string of beam crossings associated with the motion m (with $+$ denoting string concatenation).

The cost of moving from state s to $\hat{s}$, $c(s, \hat{s})$, could be a function of the length of the motion m, the risk associated

with moving to state $\hat{s}$, or any arbitrary cost function depending on the problem being solved. One should note that the beam-graph is infinite, owing to the fact that the string $b(s)$ for a state s can be of arbitrary length. However, if the graph is constructed incrementally by interleaving construction and search, we can adopt a variety of pruning strategies (Sec. IV-E) to ensure that the graph search is tractable.

A summary of the successor function used to generate the graph is given in Algorithm 1. The $ApplyRelations$ function is described in more detail in Sec. IV-C.

Algorithm 1

GetSuccs(s)

$S = \{\}$
for all motion $m \in M(s)$ **do**
  if m is a valid motion starting at s **then**
    $\hat{x} =$ configuration of m applied to $x(s)$
    $\hat{b} = b(s) + w(s, m)$
    $\hat{b} =$ ApplyRelations($\hat{b}$)
    if $\hat{b}$ is a valid string **then**
      $\hat{s} = \{\hat{x}, \hat{b}\}$
      $S = S \cup \hat{s}$
    end if
  end if
end for
return S

C. String Relations

As observed previously, we can plan with different kinds of topological constraints by considering some beam crossing strings to be equivalent under different relations. Many such relations can be expressed in terms of equivalent substrings. For example, we might say that the substrings cc' and $c'c$ are equivalent to the empty substring; this would imply that beam crossings in opposite directions cancel each other. This property is one intuitive characteristic of a *winding* constraint: if we wind a string five times around a pole in one direction, and then wind five times again in the opposite direction, the string is effectively wound zero times around the pole.

If we apply just this equivalence relation in $ApplyRelations$, then it can be shown that there exists a bijection between equivalence classes of strings under this relation and homotopy classes of paths with fixed endpoints; i.e., one path can be continuously deformed to another iff. they have equivalent strings under this relation. If we further introduce the substring equivalence $ab \sim ba$, a bijection exists between the equivalence classes and homology classes of paths with fixed endpoints; i.e., two paths have equivalent winding angles for all ROI iff. they have equivalent strings under this relation. We will elaborate on this point shortly.

Finally, we can plan for a spectrum of topological constraints consisting neither of homology nor homotopy constraints by defining other equivalence relations. Table I shows some different possibilities.

D. Equivalence of String Relations and Topological Constraints

A particularly interesting topological space X for robot navigation is the multiply punctured plane $X = \mathbb{R}^2 \backslash \mathcal{O}$, $\mathcal{O} \subset \mathbb{R}^2$. This is simply the plane $\mathbb{R}^2$ with $|\mathcal{O}|$ points removed from it (thereby the name "multiply punctured"). One can think of the multiply punctured plane as the robot's environment, with the holes corresponding to ROI in the environment. The *fundamental group* or *first homotopy group* $\pi_1(X)$ [11, 7, 10] for a topological space X is the set of all *loops* (paths with identical start and end points) based at some $x_0 \in X$, defined over the concatenation operation.

The fundamental group $\pi_1(X)$ for the multiply punctured plane with n holes is the *free group on n letters*, F_n. The free group is so called, because it can be constructed using only a set of generators and the group axioms, with no other relations. For the set of generators $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, F_n is simply the set of all words that can be formed by concatenating elements of the set $\Sigma \cup \Sigma^{-1} \cup \{\varepsilon\}$, where $\Sigma^{-1} = \{\sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_n^{-1}\}$ and ϵ is the empty string. The identity element for the free group is ϵ and the concatenation operation is defined such that $\sigma\sigma^{-1} = \sigma^{-1}\sigma = \varepsilon$. A *reduced word* in F_n is a word than cannot be simplified any further by applying the group axioms.

In the terminology of [12], a *minimally sufficient collection of sensor beams* is one that forms a spanning tree on the graph of all ROI and the boundary of the environment.

Lemma 1. *For a minimally sufficient collection of n sensor beams, there exists a unique mapping from beam crossings to generators of the fundamental group F_n, under some basis representation [12].*

Theorem 2. *The construction of virtual beams described in Sec. IV-A provides a one-to-one mapping from words of $\mathcal{B}$ to elements of the fundamental group F_n for a n-punctured plane, where n is the number of ROI in the environment.*

Proof. By Theorem 1, the construction of virtual beams yields a spanning tree on the set of all ROI and the boundary. Therefore, the virtual beams form a minimally sufficient collection of sensor beams. The theorem then follows as a consequence of Lemma 1.

The correspondence between beam crossings and generators of the fundamental group completely establishes the aforementioned relations to homology and homotopy. The reader may refer to [12] for more details.

E. Pruning Strategies

As discussed in Sec. IV-B, the beam-graph can grow infinitely, making the search intractable. One way to overcome the problem is by pruning states as and when they are created. In our implementation, we use two different pruning strategies, based on the topological constraint being handled. For the homotopy constraint, a state s is pruned, i.e, declared an invalid successor, if $b(s)$ is not a *prefix* of $b(s_{goal})$. For instance, if a newly generated successor state s had $b(s) = ab'c$, and if $b(s_{goal}) = ab'de$, then s would be

pruned away. On the other hand, if $b(s_{goal})$ were $ab'cde$, s would be a valid successor. This pruning strategy has the effect of guiding the search in the exact order of beam crossings that appear in the goal string. Although this speeds up the search significantly, it has the downside of making the search incomplete—i.e, we might miss solutions that require the robot to exhibit abnormal behaviors such as crossing a beam (which is not part of the goal string) back and forth. In static environments where the beams are spaced apart by at least the length of the largest motion that the robot can take, this pruning strategy has no effect on completeness. When planning with homology constraints, a state s is pruned if $b(s)$ is not a *subsequence* of $b(s_{goal})$. The effect is the same, except that we now provide leeway in the order of beam crossings, as required by the homology constraint. Note that all pruning is performed only after applying the string relations. One can also come up with several other pruning strategies based on the length of the string $b(s)$, or tolerance in the number of "stray beam crossings" that $b(s)$ is allowed to contain.

V. Experiments

We tested our implementation of planning with beam-graphs in simulation, as well as on a real robot. As for the graph search itself, we used the A* search algorithm [6]. The heuristics for the search were computed by running a 16-connected 2D Dijkstra search from the goal to all the (x, y) cells in the environment, assuming the robot to be circular with a radius equal to the actual robot's inscribed radius. Since the 2D search essentially assumes that the robot can turn in place at no cost, the heuristic underestimates the actual cost to the goal and is hence admissible. A video that shows the results of planning for UAV surveillance in dynamic environments, and experiments on the PR2 robot is available at http://sbpl.pc.cs.cmu.edu/shared/Venkat/ICRA13/beam_graphs.mp4.

A. UAV surveillance in dynamic environments

We adapted our beam-graph for planning with topological constraints in dynamic environments, where time was a state variable. The planning was done in a (x, y, θ, b, t) 5-dimensional state space, with the variables standing for their usual meaning. We setup hypothetical surveillance scenarios, in which a UAV had to gather information about particular ROI in the environment, in the presence of hostile patrol units and static radar installations. The UAV was constrained to follow a minimum cost path, where the cost was determined by the length of the path and the risk associated with traveling too close to a radar installation. Additionally, the UAV had to completely avoid the fields of view of the moving patrol units, whose trajectories were assumed to be known. Fig. 5 depicts a scenario where the UAV follows a minimum cost loop that encloses the ROI. The figure shows two interesting behaviors— initially, the path cuts through a radar zone in order to avoid a patroller moving towards the UAV, and later, the path forms a sub-loop that serves as a waiting maneuver for the UAV to let a patroller pass before getting back to the base.

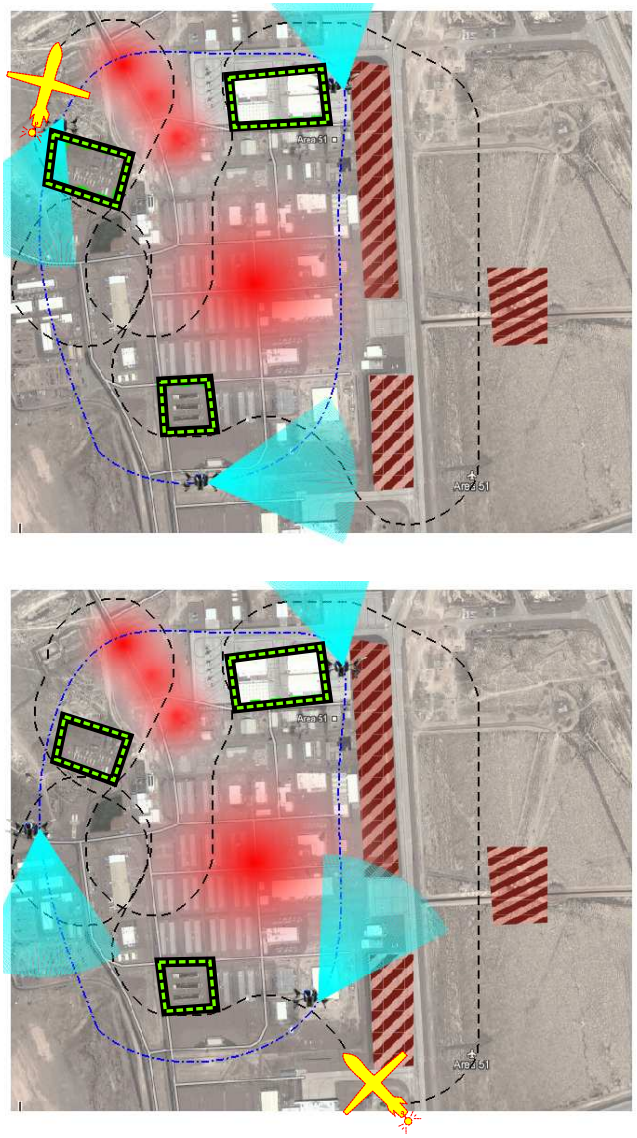
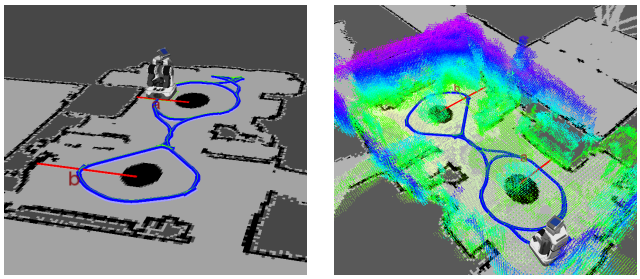


Fig. 5. UAV surveillance in dynamic environments: The UAV (enlarged and shown in yellow) is tasked with gathering information about the ROI (marked by green dashed lines) by finding an enclosing loop. The striped areas are no pass regions, the red zones are radar installations, with higher intensities corresponding to higher costs, and the blue wedges represent the fields of view of hostile patrollers that the UAV must avoid. The black dashed line is the solution path taken by the UAV.

B. Loop planning for object modeling

To demonstrate a practical application of planning with homotopy constraints, we used our planner for the task of automated 3D object modeling using a depth sensor mounted on a mobile robot. The task is to build 3D representations of objects of interest in partially structured environments. For instance, if the objects of interest are placed on isolated tables in the environment, the mobile robot would be able to build a 3D representation or point cloud by circumnavigating the tables, and then using a simple plane fitting algorithm or other sophisticated techniques to extract the individual



(a) (b)

Fig. 6. Object modeling using loop planning: (a) The red lines show the virtual beams in the environment, and the blue curve represents the path returned by the planner for the the goal string $aabb$. (b) The point cloud built using the the RGBD sensor while the robot traversed the path.

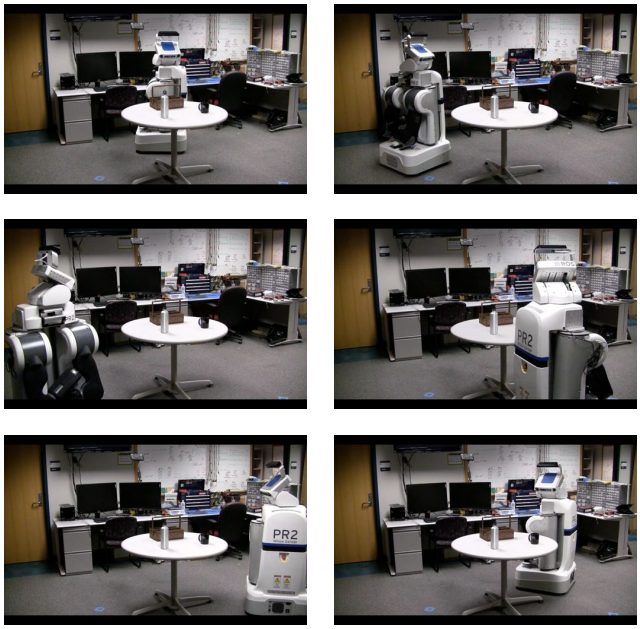
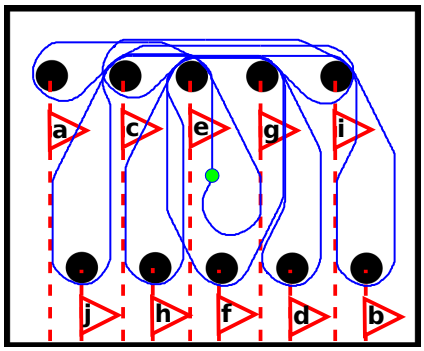


Fig. 7. The PR2 robot autonomously modeling objects on a table using a RGBD sensor, by planning with visibility constraints

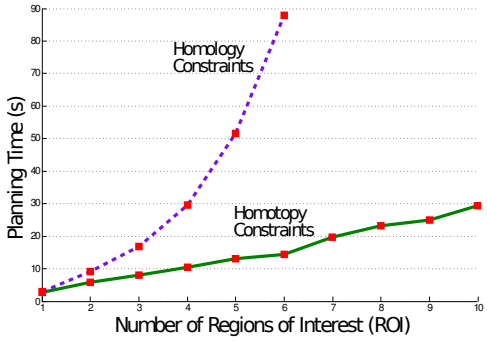
object models from the point cloud. This calls for a path that would loop around each table (or ROI) in the environment. The planning was done in a (x, y, θ, b) 4-dimensional state space, where (x, y, θ) represents the position and heading of a non-holonomic robot, and b is a string of beam crossings.

The constraint of looping around each ROI individually is the same as planning with visibility constraints discussed in Sec.II, where the goal string is composed of double counted letters from $\mathcal{B}$. For instance, consider the example in Fig. 6 where a robot has to loop around two tables in the environment. Beams a and b are constructed in accordance with the method described earlier, giving rise to the alphabet $\mathcal{B} = \{a, b, a', b'\}$. By setting the goal string $b(s_{goal}) = aabb$, we ensure that the path loops around each table at least once.

We implemented the planner in ROS (Robot Operating System) and tested it on Willow Garage's PR2 robot. The PR2 robot is a mobile manipulation platform with two 7-



(a)



(b)

Fig. 8. (a) The synthetic environment used in the experiments. The path in blue shows the solution returned by the planner for the goal string $abcdefghij$, when planning with the homotopy constraint (b) Plot showing the scaling of planning times with the number of ROI, for ordered (homotopy), and unordered (homology) beam crossing constraints

DOF mechanically counterbalanced arms, an omnidirectional base and a multitude of sensors such as stereo and high definition cameras, two lidars, and an IMU. In addition, we mounted an RGBD sensor (Microsoft Kinect) on the PR2's head, to facilitate building of 3D object models. The implementation involved writing a global planner node that plugged into the ROS navigation stack. The navigation stack provides the robot's current pose and a map of the environment to the global planner, and expects a path in return. The global planner node that we wrote also contained an additional feature for interactively selecting ROI to loop around. Additionally, we wrote a simple head behavior node that directed the PR2's head towards the nearest ROI in the environment, thereby ensuring that the RGBD sensor could obtain full 360° views of objects on the tables. Fig. 7 contains frames from the linked video that shows the PR2 robot modeling objects on tables by looping around them.

C. Computational Efficiency

To obtain insight into the computational complexity of planning with different topological constraints, we ran experiments to collect the planning times for scenarios with increasing number of ROI. The planning was done in 4D—(x, y, θ, b), and the environments used for the experiments were discretized into 230x280 cells. The orientation θ was discretized into 16 angles. We used the A* algorithm for the

graph search, and the pruning strategies described in Sec. IV-E to keep the graph tractable. Fig. 8(a) shows the synthetic environment that was used for the scaling experiments. The number of ROI was increased by successively introducing obstacles in the order corresponding to the beam names in Fig. 8(a). The goal string for each case was the concatenation of all beam crossings, i.e, the goal string was a for one ROI, and $abcdefghij$ for ten ROI. Fig. 8(b) reflects how the planning times scaled with the number of ROI. As we expected, the planning times seem to scale linearly when planning with homotopy constraints, since it imposes a strict order on the beam crossings. On the other hand, the planning times seem to grow exponentially with the number of ROI for homology constraints. This is also expected, since the algorithm has to search for the optimal solution among all possible permutations of the goal string, suggesting a factorial dependency.

VI. Conclusions

We have presented a framework for planning with a variety of topological constraints, using a graph construction called the beam-graph. Example scenarios of planning with topological constraints were discussed in the context of autonomous surveillance and object modeling through loop planning. In the domain of autonomous surveillance, we are able to enforce topological constraints to obtain full 360° views of regions of interest (ROI), and also to control the sequence of ROI surveilled. We also demonstrated the flexibility offered by our framework, by planning for surveillance in dynamic environments with time-varying costs. On the theoretical side, we proved that planning under the beam-graph framework using specific string relations is equivalent to planning with homotopy and homology constraints.

In future work, we would like to leverage more informative heuristics to make the graph search more efficient. One way of doing this could be to solve a low dimensional (x, y, b) problem and use it as a heuristic for high dimensional (x, y, θ, b) or (x, y, θ, b, t) planning problems. Additionally, we would like to investigate the notion of topological constraints in an environment where the ROI themselves are dynamic.

Acknowledgment

LaValle is supported in part by NSF grants 0904501 (IIS Robotics) and 1035345 (Cyberphysical Systems), and MURI/ONR grant N00014-09-1-1052. The other authors were partially supported by MURI/ONR grant N00014-09-1-1052.

References

[1] S. Bhattacharya, M. Likhachev, and V. Kumar. Topological constraints in search-based robot path planning. *Autonomous Robots*, 33:273–290, 2012. 10.1007/s10514-012-9304-1. I, III

[2] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proceedings Annual Symposium on Foundations of Computer Science*, pages 132–142, 1978. I

[3] S. Cabello, Y. Liu, A. Mantler, and J. Snoeyink. Testing homotopy for paths in the plane. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 160–169. ACM, 2002. I, III

[4] A. Efrat, S. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. *Computational Geometry*, 35(3):162–172, October 2006. I, III

[5] H. Gong, J. Sim, M. Likhachev, and J. Shi. Multi-hypothesis motion planning for visual object tracking. In *Thirteenth International Conference on Computer Vision*, 2011. I, III

[6] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100 –107, july 1968. V

[7] A. Hatcher. *Algebraic Topology*. Cambridge University Press, Cambridge, U.K., 2002. Available at http://www.math.cornell.edu/∼hatcher/AT/ATpage.html. II, III, IV-D

[8] E. Hernandez, M. Carreras, J. Antich, P. Ridao, and A. Ortiz. A topologically guided path planner for an auv using homotopy classes. In *ICRA'11*, pages 2337–2343, 2011. III

[9] T. Igarashi and M. Stilman. Homotopic path planning on manifolds for cabled mobile robots. In *Workshop on the Algorithmic Foundations of Robotics (WAFR'10)*, December 2010. I, III

[10] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/. IV-D

[11] J.R. Munkres. *Topology: a first course*. Prentice Hall, 1975. IV-D

[12] B. Tovar, F. Cohen, and S. M. LaValle. Sensor beams, obstacles, and possible paths. In G. Chirikjian, H. Choset, M. Morales, and T. Murphey, editors, *Algorithmic Foundations of Robotics, VIII*. Springer-Verlag, Berlin, 2009. I, III, IV-D, 1, IV-D

[13] P. Vernaza, V. Narayanan, and M. Likhachev. Efficiently finding optimal winding-constrained loops in the plane. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012. I, III