Optimal Navigation and Object Finding without Geometric Maps or Localization

Benjamin Tovar[‡] btovar@uiuc.edu
Steven M. LaValle[†] lavalle@cs.uiuc.edu
Rafael Murrieta[‡] murrieta@robotics.stanford.edu

[‡]Dept. of Electrical Engineering
ITESM CCM
Mexico City
Mexico

[†]Dept. of Computer Science
University of Illinois
Urbana, IL 61801
USA

Abstract

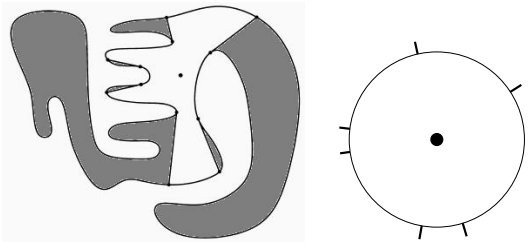
In this paper we present a dynamic data structure, useful for robot navigation in an unknown, simply-connected planar environment. The guiding philosophy in this work is to avoid traditional problems such as complete map building and localization by constructing a minimal representation based entirely on critical events in online sensor measurements made by the robot. Furthermore, this representation provides a sensor-feedback motion strategy that guides the robot along an optimal trajectory between any two environment locations, and allows the search of static targets, even though there is no geometric map of the environment. We present algorithms for building the data structure in an unknown environment, and for using it to perform optimal navigation. We implemented these algorithms on a real mobile robot. Results are presented in which the robot builds the data structure online, and is able to use it without needing a global reference frame. Simulation results are also shown to demonstrate how the robot is able to find interesting objects in the environment.

1 Introduction

Our ideas center on the development of mobile robotics systems that perform sophisticated visibility-based tasks with minimal sensing requirements. The goal is to provide autonomous mobile robots for applications such as surveillance, search-and-rescue, fire-fighting, law enforcement, and remote visual presence. Classical approaches often lack reliability when applied in practice due to classical problems such as mapping uncertainty, registration, segmentation, localization error and unpredictable control errors. A primary cause is that previous algorithmic efforts have often assumed the availability of perfect geometric models.

The guiding philosophy in our work is that most of these difficulties can be overcome by developing algorithms and mobile robots that minimize the information requirements. By constructing an algorithm and control law that use the information directly from the robot sensors, it may be possible to solve the problem while eliminating the need to make potentially-flawed measurements. The idea of using *minimal representations* was popularized in the context of manipulation planning in [6, 7]. Within mobile robotics, on-line models have been used for navigation [11, 12, 14], target tracking [8], pursuit-evasion [16], and localization [2, 4, 9, 10, 18]. By focusing on the particular task at hand, many of the classical requirements are eliminated. This can provide low-cost solutions to challenging problems, while achieving greater reliability in the face of uncertainties.

With this in mind, we present new algorithms for environment exploration, navigation, and object location. Our approach is based on a novel representation that is constructed entirely from critical events in online sensor measurements made by the robot. In this paper we introduce a dynamic tree data structure that combinatorially represents simply-connected, planar environments. The data structure serves as a topological map of the environment; however, geometric information (such as lengths, angles, distances, or segments) is not necessarily represented. We will show that once this dynamic tree is constructed, the path traversed by the robot between two locations is optimal with respect to distance, and that it is possible to find fixed objects in the environment.



(a) (b)

Figure 1: The robot's view of the environment.

2 Problem formulation

The robot is required to explore an unknown environment and to learn the location of certain objects. Once the environment is learned, the robot must be able to move efficiently and reliably between any two locations by exploiting sensor feedback. Using its learned knowledge, the robot must visit the places where *interesting* objects are, and perform useful tasks, such as moving an object from one place to another, or gathering all of the objects to one place.

To address this problem, we model the *robot* as a point that moves in a simply-connected open set, R, in the plane. The boundary of R is a simple, closed, piecewise-smooth curve. It is assumed *a priori* that the robot has no model of R.

The robot is equipped with a sensor that is capable of producing a representation as shown in Figure 1.b, which gives the location of discontinuities in depth information, for the environment shown in Figure 1.a. In a sense, Figure 1.b indicates the way the world appears to the robot at all times. Note that each discontinuity corresponds to a connected portion of R that is not visible to the robot. The precise distances to the walls may be unknown; however, it is assumed that the robot has a kind of edge detector that can detect each of the discontinuities, and return their direction relative to the robot's heading. Each discontinuity will be referred to as a *gap*, and the sensor will be called the *gap sensor* (as also considered in [16]). These gaps are referred to as *spurious edges* of the visibility polygon in [9].

It is assumed that the robot can track the gaps at all times, and record any topological change, which involves the appearance, disappearance, merging, or splitting of gaps. These changes are defined as *gap critical events*. A general position assumption is made, which states that two gap critical events do not occur at the same time.

Given the gaps the robot sees at a given time, it is possible to command the robot to move toward a given gap. This sensor-feedback movement is defined as *chasing the gap*.

Note that the path between the robot and a gap is always collision free. Also, note that the robot does not need to know its exact localization in any global reference frame to chase a gap.

3 A sensor-based dynamic tree for optimal navigation

Before discussing the structure of the dynamic tree, some concepts related to the visibility graph will be introduced. This allows us to describe the dynamic tree in a well-known framework, and to prove the optimality of the paths generated. The concept of the visibility graph is appealing for planar environments, but is difficult to implement reliably in real robots. We will show how our dynamic tree overcomes some of these implementation issues.

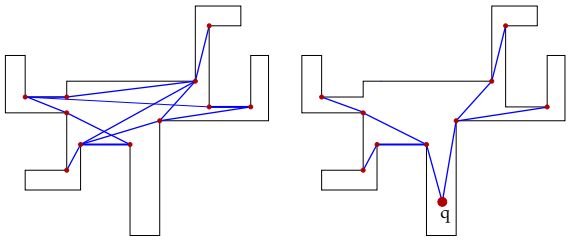
3.1 Visibility trees

Suppose that a representation of the environment is given, as a simple polygon R (Figure 2.a). For this polygon, one can compute the visibility graph, and also the *reduced visibility graph* G_{vr} [17].[1]

For any point $q \in R$, we define a *visibility tree*, $T_v(q)$, that combinatorially represents all of the shortest paths which connect q to any other point $p \in R$. See Figure 2.b. The root of T_v is q. The children of the root correspond to line segments that connect q to points of tangency along the boundary of R. The remaining edges of $T_v(q)$ are the subset of the edges of G_{vr} that are needed to construct shortest paths from q to any $p \in R$. At this point, the tree is similar to that which appeared recently in [8], which was used for on-line target tracking.

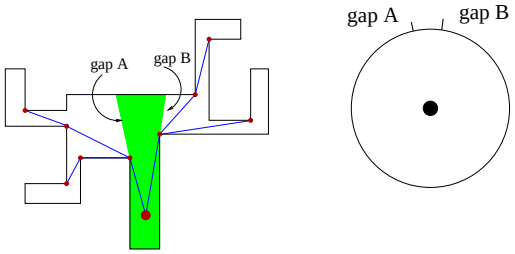
Consider the collection of possible trees, over any $q \in R$. It is possible to construct a cell decomposition of R by partitioning R based on inflection and bitangent rays extended along its boundary [9, 13]. (This is equivalent to an aspect graph based on perspective projection [15].) Let C denote any cell in R. For all points $p \in C$, $T_v(p)$ will be equivalent (the only difference is the position of the tree root); therefore, we consider these trees as a single tree, denoted as $T_v(C)$.

[1] The reduced visibility graph is obtained by eliminating the edges of the visibility graph that, if extended by a small ϵ in both sides, at least one end-point will lie outside R. Enough information is encoded in G_{vr} to produce optimal paths.



(a) (b)

Figure 2: a) Reduced visibility graph. b) Visibility tree at point q, $T_v(q)$



(a) (b)

Figure 3: a) Visibility region. b) Measurement of the gap sensor.

Suppose that an robot with perfect maps and ideal localization capability is placed in some $C \subset R$. By using the tree, $T_v(C)$, it will have sufficient information to construct an optimal path to any $p \in R$. In this paper, we assume that the robot does not have all of this information; however, the dynamic tree concept presented in Section 3.2 will enable the robot to learn and exploit this optimal-path structure without mapping and localization.

3.2 Dynamic tree definition

We next describe how to encode the shortest-path information contained in the collection of all $T_v(C)$ (for each cell C), in a single *dynamic* tree, which we refer to as T_g. The structure of T_g at a given time is similar to the corresponding $T_v(C)$, and it is updated due to critical gap events as the robot moves across cell boundaries.

From a gap sensor perspective, the gaps are produced by visibility obstructions of the vertices of R, as shown in Figure 3. Traveling to a certain gap is the same as traveling to the inflection point that produces it, which is the first part of the path encoded in $T_v(C)$. When the robot arrives at the inflection point, new gaps could appear that are to be chased by the robot. A sequence of gap chasing movements follows the same path as that represented by the visibility tree from Section 3.1. Recall that is assumed that the robot can track gaps all of the time, and record all of the critical events (appearances, disappearances, splits, and merges of gaps).

Initially, when the robot is placed in a new environment, T_g, contains only a root node, $r(T_g)$, and a set of children, one for each gap. The node $r(T_g)$ moves along with the robot, and because of this, T_g is a local topological map, not a global one as in [1, 3, 5]. In general, each leaf in T_g encodes topological information about the environment. Each child node of $r(T_g)$ represents a gap that is detected by the gap sensor while the robot is stationary in R. Assume that the children are maintained in circular order, as they appear to the gap sensor. Except for the root node, every node in T_g has a label of either "L" for left, and "R" for right. The label indicates the direction of the part of R that is hidden behind the gap. This corresponds to transitions of the gap sensor from "far to near" (left) or "near to far" (right) if the gaps are detected by a scan from right to left.

We now describe how T_g is updated as critical events occur. If a gap disappears, the corresponding node is removed from T_g. If a gap appears, it is added as a child of $r(T_g)$ in a location that preserves the circular ordering of gaps. Any node that is added in this way is designated as a *primitive node*. Intuitively, a gap disappearance means that a portion of the environment, once hidden to the robot, is now visible. An appearance means that a portion once visible is now occluded. Appearances are added to the tree as primitive nodes, meaning *already explored*. If a gap splits, then one child of $r(T_g)$ will be replaced with two children. If two gaps merge, the two corresponding children of $r(T_g)$ become the children of a new node, n, and n becomes a child of $r(T_g)$.

Formally, appearances and disappearances occur when the robot crosses inflections of R, while splits and merges occur when the robot crosses bitangents of R. Recall, from Section 3.1 that those were the borders of the set of points with a common visibility tree. The structure of T_g changes according to the cells defined for the visibility trees. Since T_g is built only from sensor readings, this eliminates the need for a geometric model of the environment, localization, and matching the tree vertices to inflection points.

3.3 Learning the dynamic tree

When the robot is placed in a new environment, all of the leaves of T_g are marked as non-primitive, since it has not yet been seen what is behind the corresponding gaps. The robot can arbitrarily choose to chase any one of the gaps, and one of the following will occur when it reaches the inflection point: 1) the gap will disappear, or 2) the gap will split. If the gap disappears, this means that the robot has seen the entire portion of the environment that was originally behind the gap. In this case, the robot must choose another non-primitive gap to chase. If the second condition occurred, and the gap splits, then the robot can choose to chase either one of the new gaps. This process can be applied repeatedly until the first condition is met and a gap finally disappears. There is one further complication. In some cases, there may be a non-primitive leaf node, n, that is not a child of $r(T_g)$. To handle this situation, the robot must first chase the child of $r(T_g)$ that is an ancestor of n in T_g. Once a split occurs, it must then follow the next ancestor. This process repeats until n finally disappears or splits.

As an example of learning T_g, suppose the robot is in the environment, as shown in Figure 4. There are seven cells; inside each cell no critical event can occur. The borders between the regions correspond to the bitangent and inflection rays. The cells are shown here to indicate when the critical events are triggered, but the robot does not know them. The root node is shown as a solid black disk. Nodes that are not known to be primitive are shown as circles, and nodes that are primitive are squares.

The robot begins to explore from Cell 1. There is only one gap which is non-primitive (Figure 4.a); hence, the robot chases it. While following this gap the robot triggers a gap split event, by crossing the border between Cells 1 and 2. This change is reflected in the tree. Remember that the root of the tree corresponds to the current location of the robot.

If the non-primitive gap that the robot is chasing disappears, then the robot must chase another non-primitive gap. The robot continues chasing the non-primitive gaps until it reaches Cell 5, at which point all leaf nodes are primitive. This condition means that the robot has seen all of the environment. The sequence of movements and trees is shown in Figure 4.

3.4 Optimal navigation

Once the environment is learned, the robot can optimally navigate to any location in the environment. Since there is no geometric map, a goal location cannot be specified in terms of coordinates. Instead, it is specified as a particular non-primitive leaf, n, that must be forced to disappear. The interpretation of this is that the actual goal is visible when the robot is in any cell that is entered into when n disappears. In this way, goals can be specified to the robot by identifying them during exploration. Later, the robot is able to return to them optimally.

Optimal navigation occurs by executing motions that follow the path in T_g from $r(T_g)$ to the desired leaf node n. During these motions, the robot maintains all changes to T_g that occur from critical gap events. At every time during the navigation, the robot chases the gap that corresponds to the child of $r(T_g)$ that is an ancestor of n. At several points, the gap may split, and the robot must chase the new child of $r(T_g)$ that is an ancestor of n. This results in a sequence of gap chasing commands, which ends when n finally disappears.

Recall from Section 3.2 and Figure 3, that following a gap is like following the edges that join vertices in the visibility tree. The paths encoded by T_g belong to a visibility tree; therefore, they are optimal. Note that although T_g encodes enough information so that the robot can reach any point in R, it is defined *locally* from the perspective of the robot. The tree T_g encodes the same information as the visibility tree $T_v(C)$, when the location, q of the robot is in C. When the robot moves and changes its cell, a gap critical event is triggered and T_g is updated accordingly. In a sense, the representation of T_g can be intuitively viewed as a visitor information map, in which the "you are here" pointer moves with the visitor, and the map always shows the shortest way to reach any location.

3.5 Finding stationary targets in the environment

The T_g data structure is not only useful for capturing the topology of the environment, but also for finding stationary objects. Even though T_g is a minimal representation, the robot can use it to solve semantic tasks, such as *go to the place where object x is*, or *move all objects to where object y is.*

Consider the example in Figure 5, which contains an interesting object which is represented a triangle. The tree T_g encodes the optimal path from the robot to the object. The events are the same as in the case of navigation, except that there are two new cases. First, when a gap is very close to an object, we associate the given object with the gap (i.e., Figure 5.b to Figure 5.c). Also, when an object associ-

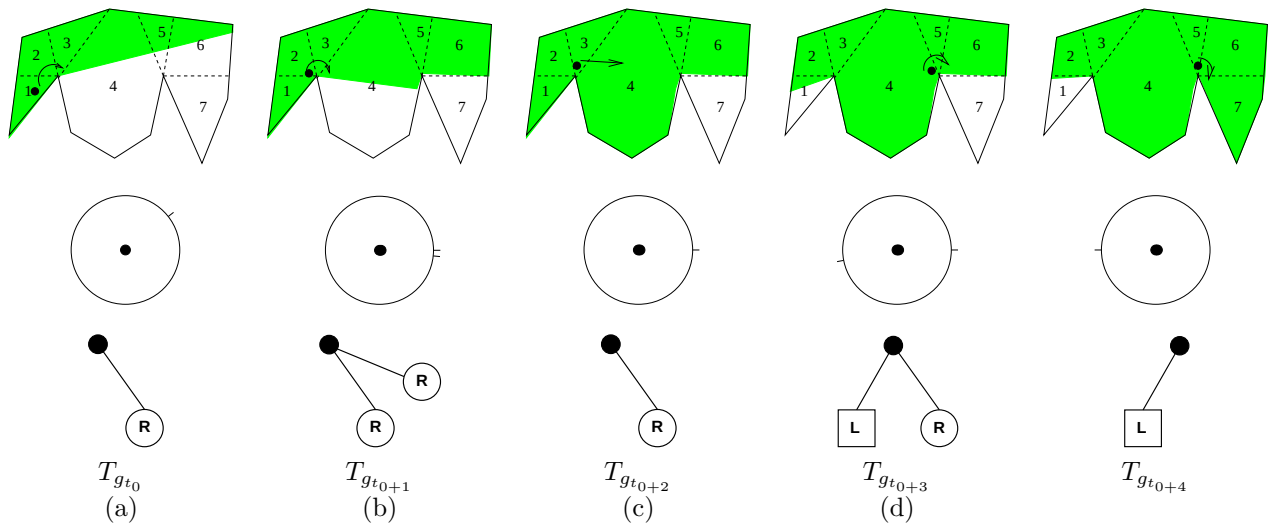


Figure 4: Learning G_g.

ated with a gap is visible, we break this association.

To locate an object, the robot follows the sequence of gaps (sequence of critical events) in T_g until it reaches the associated gap. Since the robot can travel in a straight line to the object when it is in line of sight, the path to an object from anywhere in the environment is also optimal. If the gap to which the object is associated disappears before the object is in sight, the robot can report that the object has moved.

4 Implementation and Results

4.1 Simulation

Figure 6 shows an example in which the robot was asked to explore the environment. Once it completed that task, it was asked to move all of the square objects of a certain color it might encounter to the corresponding circle of the same color. In the map the robot position is marked with a black circle, and the other circles and squares show the location of different objects. On the tree, non-primitive gaps are marked with a circle, primitive gaps with a square, gaps that hide the environment to the right with red (or dark gray) and gaps that hide to the left with green (or light gray). Figure 6.a shows the initial configuration of the robot and the corresponding partial T_g for the given environment. Note that all leaves are circles, since the robot has not done any exploration. From this point, the robot starts chasing an arbitrary non-primitive gap, while encoding into T_g the topological events of the gaps and the objects associated with them (Figure 6.b). Figure 6.c shows the tree when

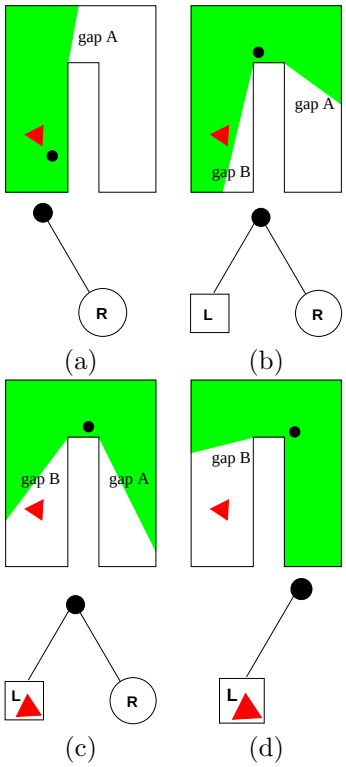


Figure 5: Encoding environment objects in G_g

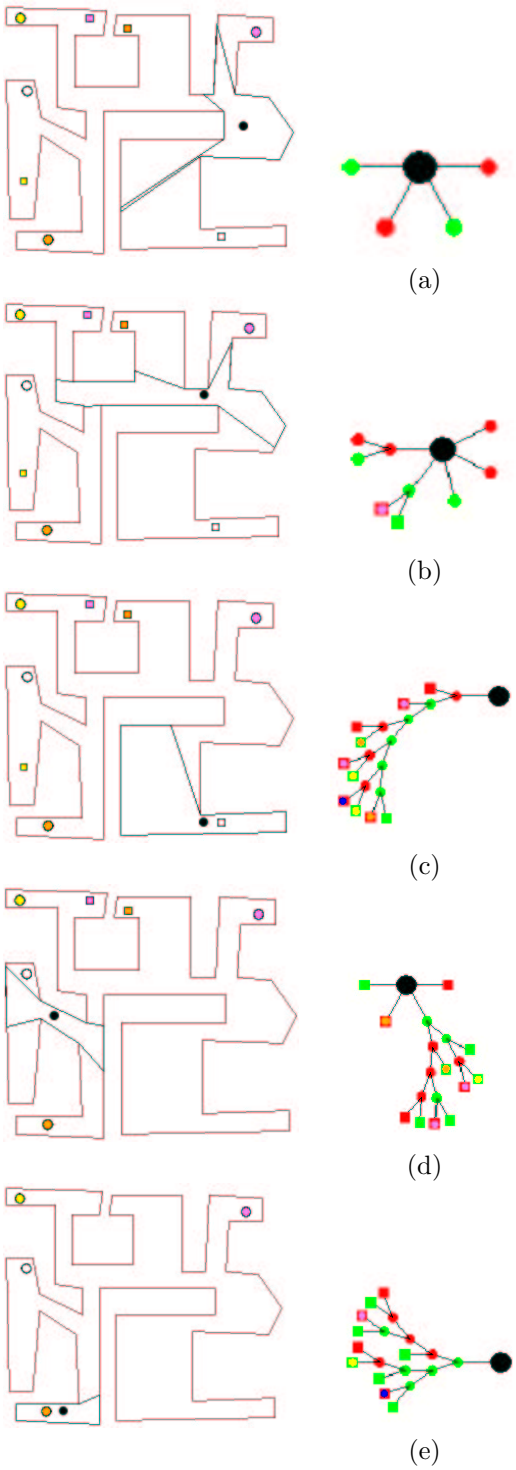


Figure 6: Dynamic tree learning and object finding simulation

the exploration is complete (note that all leaves are squares). From this point the robot begins to deliver the objects (Figure 6.d), until the task is completed (Figure 6.e).

4.2 Real robot implementation

4.2.1 Practical considerations

We implemented the T_g learning algorithm on a Pioneer 2-DX, with two SICK lasers. Although the algorithm was straightforward to implement on the real robot, there were some issues regarding the matching with the T_g robot model with the real one. The most important constraint of the T_g model is that the robot must be capable of tracking gaps all of the time. For this, the robot was equipped with two lasers to provide an omnidirectional view. The gap sensor was implemented by combining the data of these two lasers. Currently is is assumed that the gap sensor can detect discontinuities at arbitrary depth. We would like to relax this constraint in future work, but in the current setting, care was taken to have the environment walls no farther than the lasers range.

The robot motions obey the standard kinematic differential drive model. For this reason, the robot must first turn on the spot to align with the gap, and then it begins to move toward it. Note that because only gap sensor information is used, any other motion would not be guaranteed to avoid collisions.

It is worth noting that while the robot rotates to the next gap, before chasing it, it only needs to know the order of the gaps, as opposed to the precise angular position. This further enables greater robustness in the control. No global reference frame was constructed and no compass was necessary.

Because of the time required to process the gap information (laser serial communication and gap tracking) during the experiments, it was assumed that two events cannot occur during the same sensing operation (the general position assumption). In other words, the robot must move slowly enough to prevent the gap sensor from missing gap events.

The robot was also provided with wall-following capabilities. In the T_g model we considered the robot to be a point, and it can get arbitrarily close to the environment element that produces a gap. In the real robot this is not possible. When the robot gets very near to a wall, it still chases the gap, but also takes into account the information of a bumper sensor. Note that the wall-following process only assists the the gap chasing task, and is not performed in isolation.

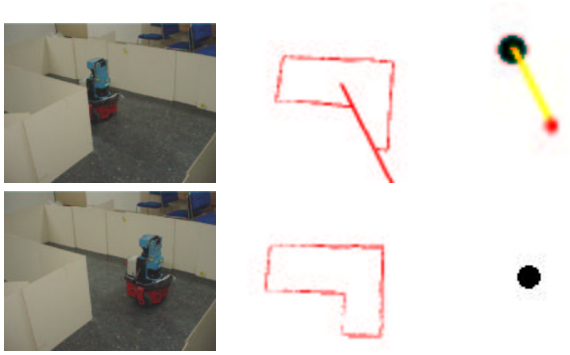


Figure 7: Disappearance example

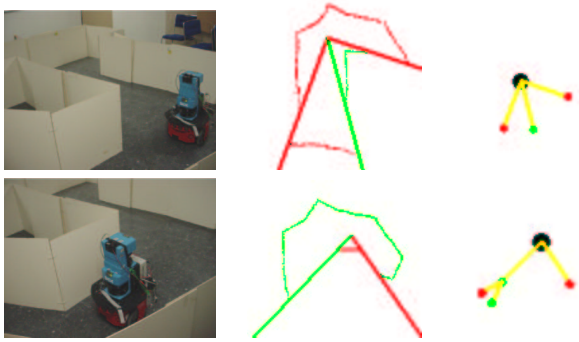


Figure 8: Merge example

4.2.2 Experiments

Figure 7 shows a disappear event. It is interesting than when the only gap disappears, the only element of the tree is the root. This condition corresponds to when the robot sees the whole environment. A merge event is shown in Figure 8, in which two gaps are merged into one when the robot hides behind a corner. Finally, Figure 9 shows some snapshots of a entire experiment, where the complete T_g of the environment is learned.

5 Conclusions

We have presented a data structure and algorithm that captures the topology of a simply-connected environment and enables a robot to navigate optimally. This data structure is a dynamic tree that encodes enough information to generate optimal paths, although only information of gap critical events is used.

Although our results to date have been encouraging, there are some issues remaining. One of our reasons of studying minimal representations is to understand what information is really needed to design robust, cost-effective robotics solutions. In the future we want to study what other capabilities should be added to the robot to relax the requirement of omni-directional, unbounded-range sensing. Also, we want to extend the work to allow multiply-connected environments.

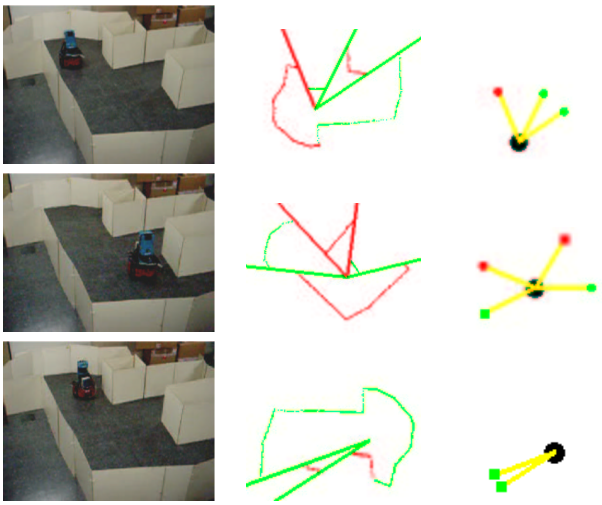


Figure 9: A complete T_g learning experiment

Acknowledgments

We thank Dr. Enrique Sucar and Marco Antonio López, of the ITESM Campus Morelos, for providing some of the equipment for the experiments. We also thank Boris Simov for helpful discussions. This work is supported in part under NSF-CONACyT Grant 0296126.

References

[1] H. Bulata and Michel Devy. Incremental construction of a landmark-based and topological model of indoor environments by a mobile robot. In *IEEE Int. Conf. Robot. & Autom.*, 1996.

[2] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. *IEEE Int. Conf. Robot. & Autom.*, 17(2), April 2001.

[3] G. Dedeoglu, M. J. Mataric, and G. S. Sukhatme. Incremental, on-line topological map building with a mobile robot". In *Proceedings of Mobile Robots XIV - SPIE*, 1999.

[4] B. R. Donald and J. Jennings. Sensor interpretation and task-directed planning using perceptual equivalence classes. In *IEEE Int. Conf. Robot. & Autom.*, pages 190–197, Sacramento, CA, April 1991.

[5] Gregory Dudek, Paul Freedman, and Souad Hadjres. Using local information in a non-local way for mapping graph-like worlds. In *Proc. Int. Joint Conf. on Artif. Intell.*, pages 1639–1645, 1993.

[6] M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE Trans. Robot. & Autom.*, 4(4):369–379, August 1988.

[7] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993.

[8] H. González-Baños, Cheng-Yu Lee, and J.C. Latombe. Real-time combinatorial tracking of a target moving unpredictable among obstacles. In *IEEE Int. Conf. Robot. & Autom.*, 2002.

[9] L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Proc. 1st Workshop on Algorithmic Foundations of Robotics*, pages 269–282. A.K. Peters, Wellesley, MA, 1995.

[10] J.B. Hayet, C. Esteves, M. Devy, and F. Lerasle. Qualitative modeling of indoor environments from visual landmarks and range data. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2002.

[11] I. Kamon and E. Rivlin. Sensory-based motion planning with global proofs. *IEEE Trans. Robot. & Autom.*, 13(6):814–822, December 1997.

[12] K. N. Kutulakos, C. R. Dyer, and V. J. Lumelsky. Provable strategies for vision-guided exploration in three dimensions. In *IEEE Int. Conf. Robot. & Autom.*, pages 1365–1371, 1994.

[13] S. M. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: An extension to curved environments. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1677–1682, 1999.

[14] V. J. Lumelsky and A. A. Stepanov. Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[15] S. Petitjean, D. Kriegman, and J. Ponce. Computing exact aspect graphs of curved objects: algebraic surfaces. *Int. J. Comput. Vis.*, 9:231–255, Dec 1992.

[16] S. Rajko and S. M. LaValle. A pursuit-evasion bug algorithm. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1954–1960, 2001.

[17] H. Rohnert. Shortest paths in the plane with convex polygonal obstacles. *Information Processing Letters*, 23:71–76, 1986.

[18] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence Journal*, 2001.