Oriented Visibility Graphs: Low-Complexity Planning in Real-Time Environments

David Wooden & Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, Georgia 30332–0250
{wooden,magnus}@ece.gatech.edu

Abstract— **We show how the introduction of a fixed goal location allows us to lower complexity compared to reduced visibility graphs. The number of inter-polygonal edges is decreased from as much as square to not more than simply twice the number of polygons. By virtue of this restriction, we demonstrate how to deploy plan-based navigation strategies in highly unstructured, dynamic environments. This approach has been exercised extensively through numerous outdoor experiments. The vehicle used was the DARPA LAGR robot, and the various test environments included trees, ditches, bushes, tall and short grass, closed canopy, and varyingly-sloped terrain.**

I. Introduction

As autonomous robots are increasingly transitioning from structured man-made environments to highly unstructured environments a number of new challenges present themselves [7], [12], [15]. In this paper we focus on the problem of navigating a mobile robot through terrain populated by a number of obstacle types: low vegetation, trees, and negative obstacles such as ditches. Information about the environment will be obtained through stereo-based descriptions of elevation and the main contribution is the production of low-complexity graphs for the purpose of path-planning over the elevation maps. In particular, we will revisit the classic visibility graph concept and reduce the size of these graphs in such a way that the interconnectivity is substantially smaller. As a result, the roadmaps may no longer be optimal (except for in special circumstances) but they will still produce solutions that are more effective than purely reactive navigation strategies.

The problem of planning efficient routes for mobile robots through dynamic environments is as old as the field of autonomous robotics itself. Two main camps have been established that can be roughly classified as *reactive* and *deliberative* (following the notation in [1]). In a reactive navigation system, the robot reacts to environmental changes in a purely local manner, i.e. not taking into account memory-based descriptions of the already-encountered environment outside the current field-of-view. This strategy enjoys wide-spread use, and its application is successful due to its inherent robustness and suitability for time-critical real-time applications. (See, for example, [2].) On the other hand, the deliberative approach can address optimality questions directly, but the computational price one is forced to pay may be too great. As a consequence, deliberate approaches have found most of their successful applications in static or slowly varying environments [10], [14].

The task which motivates this paper is that of enabling an autonomous ground robot to operate in an unknown outdoor environment in the presence of non-convex obstacles. That the robot should demonstrate improved performance upon returning to explored locations, inevitably forces the system to store and later recall information, e.g. through the generation of a map.

In fact, this work was motivated by the DARPA Learning Applied to Ground Robots (LAGR) project in which the environment is completely unknown. Moreover, not only must the robot be able to behave in a satisfactory manner, it must reach a specified goal point as quickly as possible. The robot is run through the same course multiple times, driving the need for terrain memory to improve performance.

Given that the system should be map-based, a number of existing planning solutions present themselves, including the classic visibility graphs [8], [9], cell decomposition [10], and Voronoi diagrams [11], [16]. What we require is an incremental algorithm, capable of quickly incorporating new and revised information about the environment as the robot makes its exploration. The most popular algorithm of this type is D^*, which operates on maps composed of either fixed grids [17], or more complicated space-saving structures like framed-quadtrees [18]. However, there is no visibility graph algorithm that incrementally updates connectivity and path costs between non-convex obstacles in the plane. The authors of [3] come closest, describing an algorithm for a simple polygon with a moving point (i.e. robot) on its interior, but the polygon may have no holes (i.e. obstacles) and its geometry is fixed. We present here a new algorithm adaptive to a changing set of polygonal obstacles using a modified version of the reduced visibility graph.

Since the graph is built up incrementally, the desire for optimality is outweighed by the immediate need for a feasible plan that may be only approximate. Due to the strict real-time aspect of the outdoor navigation system under consideration, complexity management is a key consideration. As such, we want to produce a path planner that exhibits the following properties:

1) The planner should always return a feasible path to the goal when one exists, given the available information;
2) The space complexity associated with the graph structure on which the planner operates should grow linearly in the problem size;

3) The running time should be kept as low as possible; and
4) The graph structure should be such that it facilitates easy human understanding.

Satisfaction of Item 2 would tend to distinguish such a planner from grid-based D^*. Where as D^*would need to store information about all areas the robot has travelled, we desire a planner that only stores information where the boundary of obstacles exists. Item 4 is important because the planner must exist within a greater system, including potentially a hierarchical control system and adaptive perception processes. Inasmuch as the planner depends on or informs with these other components, understanding how the pieces interact with each other is essential to making the system overall work as effectively as possible. In other words, the path planner does not exist in a vacuum, but must ease the interpretation of interaction amongst the other components of the robot.

The outline of this paper is as follows: In Section II the basic ideas behind the visibility graphs are recalled and an informal discussion about the possible improvements is presented. Following this, Section III describes algorithms from constructing and maintaining our novel graph structure. A brief discussion of complexity follows in Section IV. The paper concludes in Section V with elaborate, real-world experiments, including a discussion of how to transform elevation maps into polygonal obstacles.

II. Visibility Graphs

The well-known reduced visibility graph (RVG) has long been used to provide shortest-path roadmaps through a known polygonal environment [10], [13], yet, they is not typically implemented in real-time applications due to complexity issues. It takes too long to create and maintain, and consequently, quick re-planning of the graph as the environment changes is prohibitive. For convex polygons, each pair of polygons shares four edges (assuming mutual visibility), and thus the number of edges in the graph increases quadratically. Such a high number of edges negatively effects computation time (which is required for edge assignment), as well as human readability.

A. Standard Visibility Connectivity

Consider Figure 1, which illustrates the drawback of extensive connectivity in a standard reduced visibility graph. This ”spaghetti connectivity” is generated in order to provide optimal paths between any two points in the free space of the graph. Navigation between any two points is, however, not the task which we are interested in addressing in this paper. Instead, we focus on finding a good path to the known fixed goal point (i.e. we address the *single-source* problem). Hence, by orienting our approach based on this *a priori* information, the graph structure can be dramatically simplified.

B. Incremental Polygon Updates

We expect that, as the robot explores its environment, it discovers the terrain in its immediate surrounding. As it progresses, it continually updates its knowledge about the obstacles encoded in the graph by the polygons. Consider again Figure 1. If the geometry of polygon 3 (p_3) changes, should we spend our time updating edge and path costs between it and polygon 9? Given our task and limited resources, we claim the answer is no.

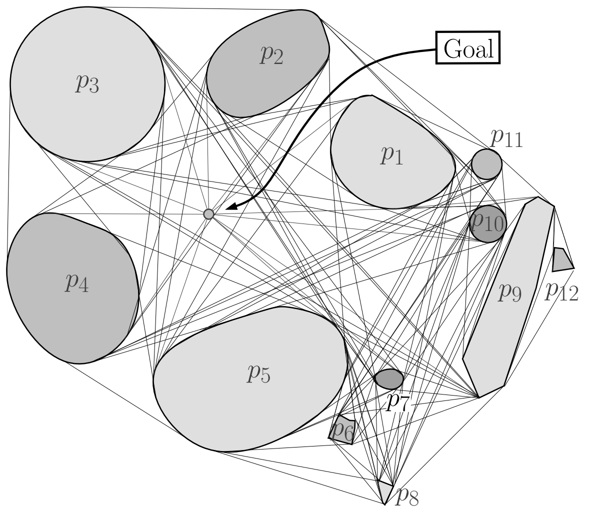

Fig. 1. Reduced Visibility Graph.

Alternatively, consider Figure 2. As the robot passes polygon 1 and changes its geometry and vertexes' path costs, it propagates a change in the polygons which are ”upstream” from it. That is, polygons 9, 10, 11, and 12 all depend on polygon 1 for their eventual connectivity to the goal and their path costs relative to it. This clear chain of dependency in the graph restricts any incremental terrain updates only to the subgraph which explicitly requires it. With this oriented perspective, there is no need to reconsider the edges of e.g. polygons 3 or 8 after modifying polygon 1. In a standard visibility graph, this limited propagation is not allowed and computation is therefore misallocated.

Note that the environment (and its polygonal representation) is constantly being adjusted by the perceptual process of our mobile robot. It is continually discovering new pieces of terrain, and revising its estimates of previously explored regions.

C. Polygon Shadows

In a planar model of the world, it is not difficult to show that polygonal obstacles cast a cone-shaped ”shadow” behind themselves, relative to a single point. In the simple example of Figure 3, an optimal path starting from a point in the shadow of polygon 1 passes by either vertex v_1^{c+} or v_1^{c-}, and optimal paths which never enter this shadow will not intersect the boundary of polygon 1. (Note that the shadows cast by polygons are not necessarily as simple as a cone based on the goal point, as illustrated by p_4 and p_5 in Figure 3.)

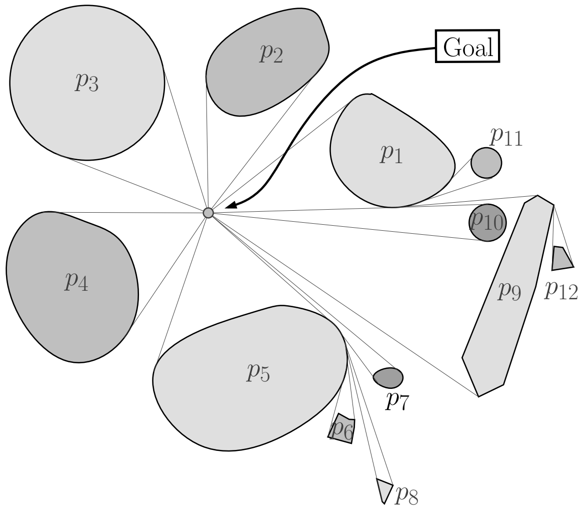

Fig. 2. Oriented Visibility Graph.

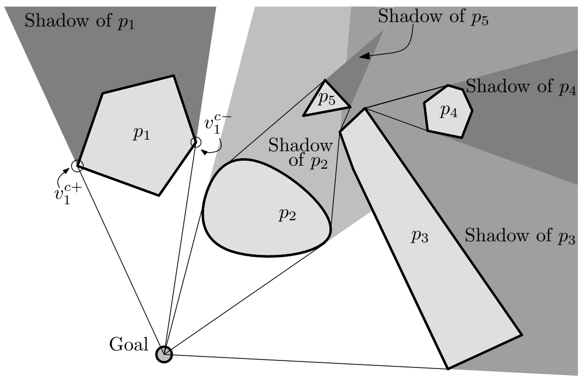

Fig. 3. Polygon Shadows.

D. Oriented Visibility Graphs

So, by leveraging our fore-knowledge of the goal location and our insight into polygon shadows, we can build our so-called Oriented Visibility Graph (OVG). By considering only the task of producing roadmaps from free-space to a goal location (i.e. *not* between any two points in free-space), we have reduced the number of edges in the graph from approximately square to not more than simply twice the number of polygons. In doing so, incremental changes in the polygonal geometry (as the robot explores the world) are localized to a subgraph, based directly on polygonal "dependencies". Hence, we expect on average less computational burden due to incremental perceptual updates about the environment, and find the readability of the graph much improved.

As illustrated by Figure 3, two edges are all that is necessary to establish the shortest path between a goal point and an point "behind" a polygon. In the applications described below, we apply this principle to non-convex polygons as well, recognizing that the resulting paths may not be optimal (as two edges are not always sufficient for shortest-paths in non-convex graphs), but accept this deficiency; the desire for optimality is outweighed by the need for a fast algorithm. And in all cases, a feasible path is always produced (when one is available).

The set of edges interconnecting polygons, E_v, grows at a much lower rate for our OVG compared to the older RVG. As stated in [9], four edges connect every pair of mutually visible convex polygons in an RVG. Hence, the cardinality of E_v for an RVG grows quadratically with the number of polygons. For an OVG, we fix the set of edges in E_v to exactly twice the number of polygons.

The main effect of the reduced size of E_v is that polygon interconnectivity is established only where necessary. When geometric changes occur in one polygon, their affect is limited to those polygons which explicitly depend on it. This is the primary complexity reduction of this approach.

III. General Polygonal Environments

In this section we present an algorithm for maintaining the oriented visibility graph.

First, we establish some notation. The graph $\mathcal{G}$ is assumed to contain a set of polygons $\mathcal{P}$ and a set of edges $\mathcal{E}$ which interconnect elements of $\mathcal{P}$. Included in $\mathcal{P}$ is p_g, the goal polygon. The total set of vertexes in $\mathcal{G}$ is $\mathcal{V}$. Let $poly(v)$ denote the polygon to which vertex v belongs, and let $vert(p)$ denote the set of vertexes of p. The source vertex and destination vertex of an edge e are denoted $src(e)$ and $dest(e)$ respectively. The robot vertex is v_r, and the goal vertex is v_g.

A. General Algorithm

The algorithm we present accepts as input an unordered list of polygon updates $\mathcal{P}_u$. That is, an element of the input list is either

1) a new polygon to be added to $\mathcal{P}$,
2) an existing polygon to be removed from $\mathcal{P}$, or
3) an existing $p \in \mathcal{P}$ with modified structure (i.e. its vertex list has changed).

Note that the algorithm in Table III-A is presented to illustrate how an OVG can be generated, and not as an example of optimally efficient implementation.

The operation $blocks(p, e)$ returns true if the boundary of p intersects e. During the last step, only those polygons whose vertexes fall within the circle centered on the goal with radius $\|v_r - v_g\|$ or within a user-defined locus of the robot are actually considered. The reason for this is that polygons outside this set are unlikely to be encountered by the robot on its quest to the goal, and are probably not worth our effort.

B. Edge Assignment

By far, the final step carries the most computational burden, but before jumping into the details, we must introduce even more notation. The function $angle(v, g, p)$ returns the angle between the vector g to v and the vector g to the center of mass of p. The visibility between points s from x is returned by $visible(x, s, \mathcal{P}_k)$, considering only polygons in $\mathcal{P}_k$. Polygons which block the visibility of two points is returned by $blockers(s, g) = \{p \in \mathcal{P} \mid p \text{ blocks } line(s, g)\}$.

For the sake of brevity and clarity, the algorithm in Table III-B is not quite complete. For example, it relies on path costs having already been computed for any polygon polled from $\mathcal{P}_{upstream}$. In the case that this assumption is violated (perhaps due to mutual path cost dependency between two non-convex polygons), polygons polled from $\mathcal{P}_{upstream}$ can be re-appended to the list and re-processed.

1) Find polygons "upstream" of $\mathcal{P}_u$:
   · Initialize a list of polygons $\mathcal{P}_{upstream}$ to $\mathcal{P}_u$.
   · `for` each $p \in \mathcal{P}_{upstream}$, `for` each $e \in \mathcal{E}$, `if` the $dest(e) \in vert(p)$, `add` $poly(src(e))$ to $\mathcal{P}_{upstream}$.
2) Prune edges that will be modified:
   · `for` each $e \in \mathcal{E}$, `if` $poly(src(e)) \in \mathcal{P}_{upstream}$, remove e from $\mathcal{E}$.
3) Remove all updated polygons:
   · `for` $p \in \mathcal{P}_u$, remove p from $\mathcal{P}$.
4) Recreate new/modified polygons:
   · `for` nonempty $p \in \mathcal{P}_u$, create vertexes $vert(p)$, add to $\mathcal{V}$, and add p to $\mathcal{P}$.
5) Remove blocked edges of unmodified polygons:
   · `for` $p \in \mathcal{P}_u$, for each $e \in \mathcal{E}$, remove e if $blocks(p, e)$.
6) Sort the modified polygons and reset each the path cost of each vertex in these polygons:
   · `sort` $\mathcal{P}_{upstream}$ by $\underline{d}_g(p)$, $\forall p \in \mathcal{P}_{upstream}$.
7) Add edges and calculate path costs for all upstream polygons.

TABLE I
ORIENTED VISIBILITY GRAPH ALGORITHM.

```
forall $p \in \mathcal{P}_{upstream}$
    set $g = v_g$
    find the shadow casting vertexes:
    $v_p^{c+} = \arg\max_{v \in p}(angle(v, g, p))$
    $v_p^{c-} = \arg\min_{v \in p}(angle(v, g, p))$
    forall $s \in \{v_p^{c+}, v_p^{c-}\}$
        $P_b = blockers(s, g)$
        do
            forall $q \in \mathcal{P}_b$
                $w = \{x \in q \mid visible(x, s, \mathcal{P}_b)\}$
                $v_q = \arg\min_{x \in w}(\mathcal{C}(s) + \mathcal{W}(x, s))$
                append $v_q$ to $\mathcal{V}_q$
            end
            $g = \arg\min_{v \in \mathcal{V}_q}(\mathcal{C}(v))$
            recompute the shadow caster $s$
            recompute $\mathcal{P}_b = \mathcal{P}_b \cup blockers(s, g)$
        until $s$ and $\mathcal{P}_b$ stabilize
        add the edge between $s$ and $g$
    end
    assign path costs for $v \in \mathcal{V}_p$
end
```

TABLE II
OVG EDGE ADDITION ALGORITHM.

IV. Complexity

The worst-case running time for this algorithm is $O(|\mathcal{V}|^3)$. The average complexity (based on experiments described below), however, is much lower. Moreover, this algorithm does make use of any known advanced methods for sorting and searching vertexes and detecting collisions (e.g. the *kd-tree*, the radial sweep principle [4], or the funnels of [5]). Both algorithms are presented as they are to be clear though inefficient approaches to producing Oriented Visibility Graphs. The benefit that the OVG offers is that incremental changes in the graph polygons propagate edge and path cost modifications to a confined subgraph. Hence, while average running time as a function of input size for our approach may not be better than of some RVG implementations, the average input size to the OVG is much much lower.

The edges of the OVG are a subset of the RVG edges. As discussed above, space requirements for an OVG's edges grow only linearly in the number of polygons. While algorithmic complexity reduction is not the focus of this paper, the OVG can be considered low-complexity by virtue of the small number of polygonal interdependencies. Our use of this graph approach on DARPA's LAGR project bears out that the complexity of the graph is well managed.

V. Application - Proof of the Pudding

In this section, we describe in brief how we have applied our Oriented Visibility Graph to the navigation of a ground robot that is given a fixed goal point in GPS coordinates, and through a GPS receiver knows approximately its own location. It is expected to traverse the outdoor environment and reach the goal in as little time as possible. Information about its surroundings is gathered through only four small cameras (two stereo pairs) and a bump sensor. These cameras produce stereo maps of 4-6 meters in maximum depth, which are used in turn to accumulate an elevation map of the terrain. Also, a traversability map is produced from the raw images and combined with elevation to determine where the robot may travel to reach the goal.

Navigation commands are provided to the robot through a planner based on an OVG. This planner listens to the elevation and traversability map datastreams, and updates its graph accordingly.

The two stereo pairs generate stereo disparity maps at 4Hz each. This information runs through the process described in Section V-A and polygon updates are handed to the planner. The planner directly produces motor commands for the robot, and runs between 4 and 20 Hz. (The robot *must* receive motor commands at above 2Hz or otherwise behaves undesirably.) Our planner's average cycle time is above 5Hz.

The robot has been tested in outdoor courses with total distances over 100 meters, in open terrain, on paths through woods, and under tree canopy without trails. It is given three runs to attempt the same course, starting from about the same location. At the end of the first and second run, the robot saves its graph so that it may be re-loaded at the initiation of the second and third runs. It is the planner's job to find its way out of cul-de-sacs as it discovers them, and avoid them all together if it returns to them. The outdoor environment contains both natural and man-made cul-de-sacs and non-convex polygons to challenge the robot.

A. *Generating Polygons from Sensor Data*

We consider only data streams that correspond to Cartesian image maps, inasmuch as our visibility graph is presented here as strictly 2-D. With each stream, the graph is informed about the likely presence of some object type at a specific location, and the variance estimate of that likelihood. These multiple likelihood and variance maps are subsequently combined via a function $c(...)$ into a single likelihood-of-obstacle image L. An example of c, appropriate for the block diagram in Figure 4, is

$$c(s, v_s, t, v_t) = \begin{cases} s, & \text{if } v_s < \theta_{v_s},\, t < \theta_t,\, v_t < \theta_{v_t} \\ 0, & \text{otherwise,} \end{cases}$$

where s refers to a sort of first derivative of elevation, t is the computed traversability computed for a pixel location, and v_s and v_t refer to the variance of measurements of s and t. The various θ_i refer to user-defined threshold parameters for t, v_t, and v_t. We compute s at a pixel location by the well-known sobel operator [6].

Now, L must be transformed via a binary decision-making function $d(L(i,j))$, identifying which pixels the robot can traverse. The simplest non-trivial decision function is naturally

$$d(L(i,j)) = \begin{cases} \mathit{obstacle}, & \text{if } L(i,j) > \theta_l \\ \text{not } \mathit{obstacle}, & \text{otherwise} \end{cases}$$

where θ_l is some threshold on L.

Hence, let T be the mapping from the real-valued map L to the binary obstacle map M,

$$T : \Re^{nxm} \rightarrow \mathbb{B}^{nxm}. \quad (1)$$

By applying d at each location of L, we transform L into M.

Obstacle points in M are segregated and labelled based on any typical segmentation technique.

Of course, all the operations in Figure 4 are incremental. So, updates are passed in the form of individual pixel modifications, and operations like segmentation are performed on a pixel-by-pixel basis.

B. *Polygonization*

The labelled obstacle map of Figure 4 is polygonized for input to the OVG-based planner. Polygonization is performed according to the following steps:

1) A mathematical-morphology dilation operation with a circular structuring element is applied to the labelled obstacle points for each obstacle (e.g. Figure 5). This dilation accommodates the physical geometry of the robot, allowing it to maintain an appropriate distance between it and obstacles.
2) By starting at any point on the boundary of the dilation from Step 1, the closed-contour set of pixels can be generated by iteratively stepping from one pixel to the next.
3) The boundary walk of Step 2 produces more pixels than necessary to accurately represent the obstacle; a reduction of these vertexes can be performed. Let δ_i be distance from a vertex v_i to the line formed by its two neighbors along the boundary. By removing those vertexes with δ less than some threshold, a representation of the obstacle is found which has fewer vertexes. Of course, fewer vertexes per polygon implies decreased running time, but tends to misrepresent the obstacles that the robot is to avoid.

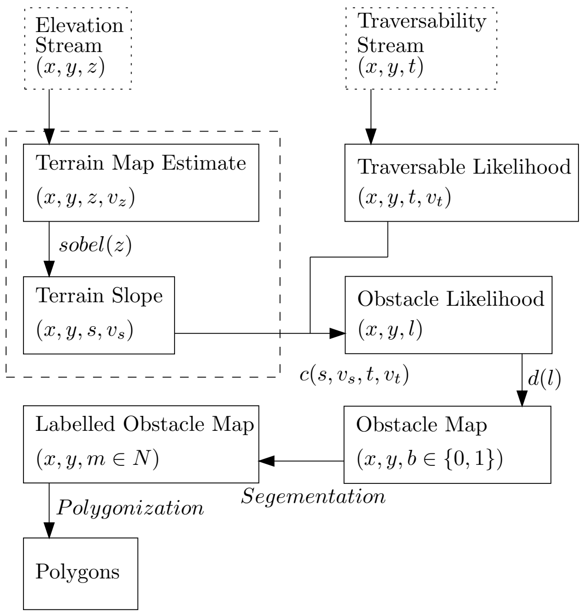
Elevation Stream (x,y,z)
Traversability Stream (x,y,t)
Terrain Map Estimate (x,y,z,v_z)
$sobel(z)$
Terrain Slope (x,y,s,v_s)
Traversable Likelihood (x,y,t,v_t)
Obstacle Likelihood (x,y,l)
$c(s,v_s,t,v_t)$
$d(l)$
Labelled Obstacle Map $(x,y,m\in N)$
Obstacle Map $(x,y,b\in\{0,1\})$
$Segementation$
$Polygonization$
Polygons

Fig. 4. Traversability Streams to Polygons.

C. *Samples from Application*

Images such as in Figure 6 are used to form stereo disparity maps and the elevation stream for Figure 4. Figure 7 illustrates the graph structure overlaid on the elevation map of a test run. Polygons are shown in white, and edges are black. Here, each pixel represents a $0.1m \times 0.1m$ square. Graphs typically contain as many as 100 polygons of various sizes, are composed of thousands of vertexes, and cover more than 100 meters from start to finish. Even with the naive Algorithms presented above, the planner still operates fast enough for our real-time system.

VI. Conclusions

In this paper, we derive a significantly reduced roadmap for unstructured polygonal environments compared to the reduced visibility graph. This construction stems from the

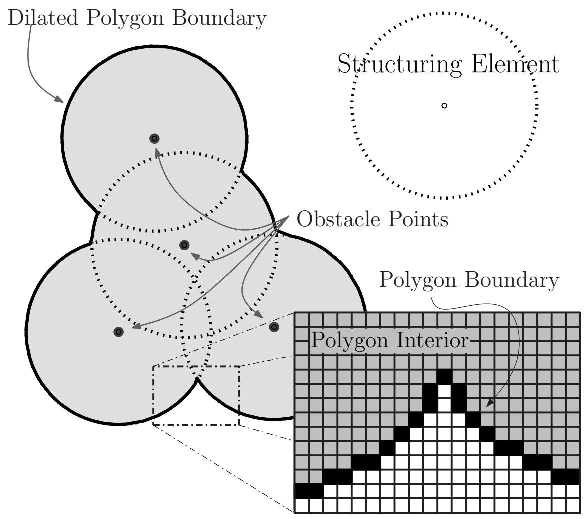

Fig. 5. Sample Polygonization with Circular Structuring Element.

Fig. 6. Sample Stereo Images.

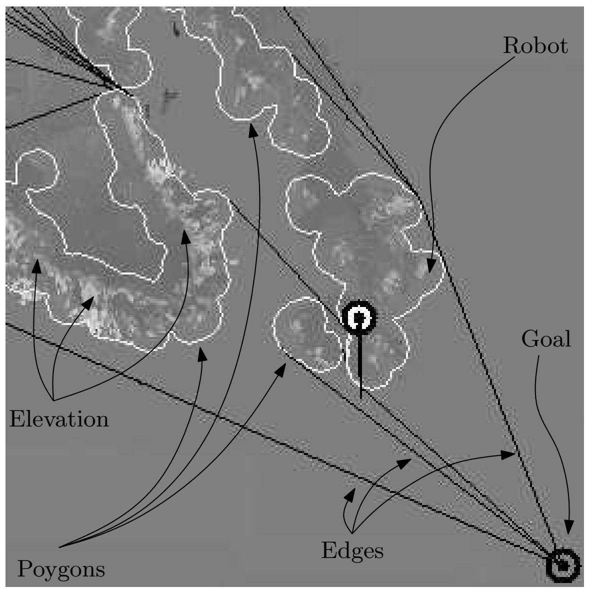

Fig. 7. Sample Terrain and Graph.

fact that we insist on a given, fixed goal point. Real-world experiments illustrate the usefulness of the proposed method in time-critical outdoor applications where the perception is based solely on stereo-based elevation maps. These maps are polygonized in order to support the use of the planner. By saving the graph between runs, dynamic update rules (for adding, removing, or changing polygons) enable the robot to improve its performance over runs.

Acknowledgements

The authors would like to thank Tucker Balch for his insightful discussions on the planner, and the Georgia Tech LAGR team for providing a research platform on which this work was implemented and tested.

REFERENCES

[1] Arkin R.C., Behavior-Based Robotics, *The MIT Press*, Boston, MA, 1998.

[2] -, *Navigational path planning for a vision-based mobile robot*, Robotica, v 7, pt 1, p 49-63, Jan. 1989.

[3] B. Aronov, L. J. Guibas, M. Teichmann, and L. Zhang. Visibility queries in simple polygons and applications. *International Symposium on Algorithms and Computation*, pp. 357-366, 1998.

[4] de Berg M., M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer, Berlin, 1997.

[5] Ghosh S.K., D. M. Mount, *An output sensitive algorithm for computing visibility graphs*, SIAM J. Comput., v 20, p 888-910, 1991.

[6] Gonzalez R.C., R.E. Woods, Digital Image Processing, *Addison-Wesley*, 1992.

[7] Hebert M., A. Stentz, C. Thorpe, *Mobility Planning for Autonomoun Navigation Multiple Robots in Unstructured Environments*, IEEE International Symposium on Intelligent Control - Proceedings, p 652-657, 1998.

[8] Huang H.-P., S.-Y. Chung, *Dynamic Visibility Graph for Path Planning*, Proceedings 2004 IEEE/RSJ International Conference on Itelligent Robots and Systems, pt 3, v 3, pp 2813-18, 2004.

[9] Latombe J.-C., *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[10] LaValle S.M., *Planning Algorithms*, Cambridge University Press, To be published 2006.

[11] Lee D.T., R. L. Drysdale, *Generalization of Voronoi diagrams in the plane*, SIAM J. Computing, v 10, pp 73-87, 1981.

[12] Nieto J., J. Guivant, E. Nebot, S. Thrun, *Real time data association for FastSLAM*, IEEE International Conference on Robotics and Automation, pt 1, pp 412-18, v 1, 2003.

[13] Nilsson N.J., *A mobile automaton: An application of artificial intelligence techniques*, 1st International Conference on Artificial Intelligence, pp 509-520, 1969.

[14] Oommen B.J., S.S. Iyengar, N.S.V. Rao, R.L. Kashyap, *Robot navigation in unknown terrains using learned visibility graphs. Part I: The disjoint convex obstacle case*, IEEE Journal of Robotics and Automation, v RA-3, n 6, pp 672-81, Dec. 1987.

[15] Rezaei S., J. Guivant, E.M. Nebot, *Car-like robot path following in large unstructured environments*, Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, pt 3, pp 2468-73, v 3, 2003.

[16] Sharir M., *Algorithmic motion planning*, In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, 2nd Ed., pages 1037-1064. Chapman and Hall/CRC Press, New York, 2004.

[17] A. Stentz. Optimal and efficient path planning for partially-known environments. *Proc. IEEE Int. Conf. Robot. & Autom.*, pages 3310-3317, 1994.

[18] A. Yahja, A. Stentz, S. Singh, B. Brumitt, Framed-Quadtree Path Planning for Mobile Robots Operating in Sparse Environments. *Proc. IEEE Int'l Conf. on Robotics and Automation*, pp. 650-655, 1998.