Robot Localization: Theory and Implementation*

Oliver Karch Hartmut Noltemeier Thomas Wahl

Department of Computer Science I
University of Würzburg
Am Hubland, D–97074 Würzburg, Germany
Email: {karch, noltemei, wahl}@informatik.uni-wuerzburg.de

1 Introduction

We consider the *robot localization problem* described as: a robot is at an unknown position in an indoor-environment and has to determine where it is located. This problem occurs if, for example, the robot "wakes up" after a breakdown (e.g., a power failure) and knows nothing about its initial configuration before the breakdown.

We assume that the robot has a *polygonal map* $\mathcal{P}$ of its environment, a range sensing device (e.g., a laser radar), which allows it to determine its *visibility polygon* $\mathcal{V}$, and a compass (i.e., the robot already knows its orientation). Then, the localization process usually consists of two stages:

First, the non-moving robot *enumerates all hypothetical positions* that are consistent with its sensor data, that is, all points $p \in \mathcal{P}$ such that the visibility polygon $\mathcal{V}_p$ of p equals the measured polygon $\mathcal{V}$. For this task, Guibas, Motwani, and Raghavan [2] present a simple algorithm with running time $\mathcal{O}(nm)$ if $\mathcal{P}$ consists of n and $\mathcal{V}$ consists of m vertices. They also propose a more sophisticated data structure to perform this query in time $\mathcal{O}(\log n+m+k)$ with preprocessing costs of $\mathcal{O}(n^4 r)$ in the worst case, where r is the number of reflex vertices of $\mathcal{P}$ and k is the number of hypothetical robot positions, for which $k \leq r$ can be shown.

If there are several robot positions that induce the same visibility polygon, they cannot be distinguished by a non-moving robot. In those cases, the robot *eliminates the wrong hypotheses* and determines its position exactly by moving around in its environment (as little as possible) in the second stage. For this, Dudek, Romanik, and Whitesides [1] describe a simple competitive greedy strategy with time and space complexity $\mathcal{O}(k^2 n^4)$, which was recently improved by Schuierer [5] to a running time of $\mathcal{O}(kn \log n)$ and space complexity of $\mathcal{O}(kn)$.

In this talk we concentrate on generating the possible robot locations in the framework of the method proposed by Guibas *et al.* We present a slightly sharper bound of $\mathcal{O}(n^2 r(n + r^2))$ for the preprocessing costs, in which the dependence on the number of reflex vertices is more evident.

Since this method requires an exact map and error-free sensors it is not directly applicable in practice. To overcome this problem we are currently working on the use of *distance functions* to model the resemblance between the range scans and the preprocessed skeletons. We have already implemented a variant of the original method of Guibas *et al.* (with some simplifications) and are going to adapt it in order to test different distance functions for their usability in practice.

2 A Sharper Preprocessing Bound

2.1 Preliminaries

The main idea in [2] is to decompose the map $\mathcal{P}$ into *visibility cells* such that all points p of one cell $\mathcal{C}$ have identical *visibility skeletons* $V_p^* = V_{\mathcal{C}}^*$. That is, their visibility polygons $\mathcal{V}_p$ are essentially the same and differ only in spurious edges (collinear with the viewpoint p) or in those map edges that are only partially visible (due to occlusions by reflex vertices). In the corresponding skeleton, such a partially visible edge is replaced by an *artificial edge* together with the line equation of the original edge.

A position where a skeleton V^* "fits" into the map is called an *embedding* $h(V^*)$ of the skeleton. Figure 1 shows an example: the skeleton V^* on the left has one artificial edge a_1 (with the corresponding line g_1). There exist three embeddings $h_1(V^*), \ldots, h_3(V^*)$ such that the skeleton fits into the map. The visibility cells with skeleton V^* (dark gray) are those places where exactly one of the three *candidate edges* $e_1, \ldots, e_3$ is (partially) visible through the embedded artificial edge $h_i(a_1)$. These candidate edges are collinear and lie on the embedded line $h_i(g_1)$.

*This research is supported by the Deutsche Forschungsgemeinschaft (DFG) under Proj.no. No 88/14-1.

For the localization process, it can be shown that, in order to determine all points that have $\mathcal{V}_p$ as their visibility polygon, it suffices to consider all embeddings $h_i(V_p^*)$ of the skeleton V_p^*, and then to check whether the corresponding embedded viewpoint $h_i(p)$ induces the same skeleton V_p^*. (That is, the point $h_i(p)$ must lie in a visibility cell $\mathcal{C}$ with $V_\mathcal{C}^* = V_p^*$.)

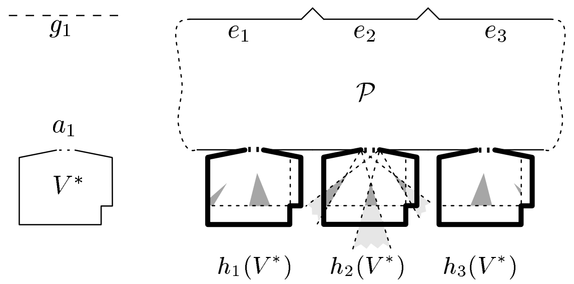

Figure 1: A skeleton V^* with three embeddings (bold) $h_1(V^*), \ldots, h_3(V^*)$. The three visibility cells (dark gray) in embedding $h_2(V^*)$ are created by intersecting the visibility wedges (light gray) with the kernel $\mathrm{Ker}(h_2(V^*))$.

Using this idea along with some sophisticated data structures and algorithms, Guibas *et al.* are able to show that for map polygons without holes the localization problem can be solved with a query time of $\mathcal{O}(\log n + m + k)$ and preprocessing costs of $\mathcal{O}(n^2 r \cdot |\mathcal{EC}|)$.

To this end, the visibility cells are divided into *equivalence classes* according to their skeleton, and $|\mathcal{EC}|$ denotes the worst case-complexity of those equivalence classes. The complexity of such a class $\mathcal{EC}_{V^*}$ is simply the total number of vertices of all visibility cells in $\mathcal{EC}_{V^*}$ (i.e., the number of vertices of all cells with skeleton V^*). In [2] this complexity is shown to be in $\mathcal{O}(n^2)$.

2.2 A Sharper Bound for $|\mathcal{EC}_{V^*}|$

To establish a sharper bound of $\mathcal{O}(n + r^2)$ for the complexity of any equivalence class $\mathcal{EC}_{V^*}$, we have to count carefully the vertices/edges of the visibility cells that have V^* as their skeleton. To this end, we show that each such visibility cell (which lies in an embedding $h(V^*)$ of V^*) is bounded by only two different types of edges:

- edges of the *kernel* $\mathrm{Ker}(h(V^*))$,
- edges of *visibility wedges* which are induced by the candidate edges for the (embedded) artificial edges of the skeleton.

For an artificial edge $h(a)$ in an embedding $h(V^*)$ and a candidate edge e, the corresponding visibility wedge is the set of all points for which only points of the candidate edge e are visible when looking "through" the artificial edge $h(a)$.

Figure 1 illustrates this: each of the three candidate edges $e_1, \ldots, e_3$ induces a visibility wedge in the embedding $h_2(V^*)$. The union of the three wedges for the artificial edge $h_2(a_1)$ intersected with the kernel $\mathrm{Ker}(h_2(V^*))$ yields the set of visibility cells in $h_2(V^*)$ that are equivalent to V^*.

It is easy to see that the number of edges of the second type, which depends strongly on the total number of candidate edges, dominates the complexity of $\mathcal{EC}_{V^*}$. Therefore, let $\overline{E}$ be the set of *all* candidate edges for *all* artificial edges in *all* embeddings of the skeleton V^*. For map polygons without holes or with only convex holes, the cardinality of $\overline{E}$ is in $\mathcal{O}(r)$.

If each candidate edge of $\overline{E}$ could be assigned to at most one visibility wedge, the complexity of the arrangement of all wedges, $\mathcal{O}(|\overline{E}|^2) = \mathcal{O}(r^2)$, would be an upper bound for the considered number of edges. But, unfortunately, this is not possible as is depicted in Figure 1: edge e_2 induces a visibility wedge in all of the three embeddings. Therefore, edge e_2 cannot be uniquely assigned to one of the visibility wedges and this naive approach does not succeed.

But, with the observation that each *pair* of candidate edges can be assigned to at most one *pair* of visibility wedges, we are able to show the following theorem. (We omit the details in this context; the complete proof is given in [3].)

Theorem 1 *The total complexity of any equivalence class for a map polygon without holes is in* $\mathcal{O}(n + r^2)$.

This result can be extended to map polygons with only convex holes. For this setting, it can be shown that the complexity of $\mathcal{EC}_{V^*}$ is in $\mathcal{O}(n + (l+1)r^2)$, where l is the number of holes.

3 Implementation of the Algorithm

In order to investigate the behavior of the algorithm, we implemented a simplified version of it in C++. "Simplified" means that we did not focus our efforts on observing data structures and programming details that were suggested by Guibas *et al.* Rather, we wanted to have an instrument to *exper-*

iment with different inputs for the algorithm, as to make it reasonably stable to be used in real-life environments in the future. A major consequence is that the program does not keep to theoretical time and space bounds proven in [2, 3], as this would have required tremendous programming efforts. The program correctly handles any topological special cases, in particular self-similarities in the scene which may occur when the obstacles or the border polygon have many collinear or parallel edges.

Now let us briefly describe the course of the program. In a first step, the visibility cell decomposition is computed by introducing visibility lines into the scene. This decomposition defines a planar straight line graph, which partitions the plane into regions; we distinguish between the outer region, visibility cell regions, and obstacle regions.

Next, the visibility skeleton for each cell region is computed and stored in an incremental fashion by walking from one cell to one of its neighbors. (At this point, we take up a lot more space and time than it was granted by Guibas *et al.*) This concludes the preprocessing.

In the query, the skeleton of the input point is determined from the visibility polygon and compared to the skeletons of all visibility cells. (In particular, we currently do not search for matching skeletons in a sophisticated tree data structure as proposed in [2].) Whenever a match is found, the corresponding cell is highlighted. We now overlay the query skeleton with any matching cell skeleton. Such we obtain the map position of the query point that is induced by the cell skeleton. If this point is inside the cell, an answer to the query is found and marked as a "solution".

All constructive phases of the algorithm, as the cell decomposition, can be processed in a step-by-step manner, that is, the user may check intermediate results at any point. The program also includes a demonstration of the visibility polygon/skeleton computation, which can be used to explain the idea of the algorithm.

For many data structures, we used the LEDA Library of Efficient Data Types and Algorithms [4]. This includes all basic geometric objects, the planar straight line graph and others. Most of LEDA's data structures are provided also for rational coordinates in order to achieve exact arithmetic. The use of LEDA's robust algorithms (e.g., for the computation of line arrangements) is another reason not to focus on (theoretically) optimal running time or space requirements, since in that case many of these algorithms would have had to be modified or could not have been used at all.

4 Usability in Realistic Scenarios

When we try to use the localization method described above in realistic scenarios, the visibility skeleton that the robot computes from its sensor data usually does *not exactly match* any of the preprocessed skeletons due to *sensor uncertainties*. That is, the robot is not able to determine the correct equivalence class of skeletons and the localization process completely fails.

Our suggestion to tackle this problem is to use an appropriate *distance function* $d(S, V^*)$ that models the *resemblance* between a range scan S (from the sensor) and a preprocessed skeleton V^*. Then, a localization query corresponds to a *nearest neighbor query* in the set of skeletons with respect to the chosen distance function $d(S, V^*)$.

Currently, we are investigating the usability of different distances theoretically, and are going to implement and test some distance functions in the framework of our localization software described above.

References

[1] G. Dudek, K. Romanik, and S. Whitesides. Localizing a Robot with Minimum Travel. In *Proceedings of the 6th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 437–446, 1995.

[2] L. J. Guibas, R. Motwani, and P. Raghavan. The Robot Localization Problem. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Algorithmic Foundations of Robotics*, pages 269–282. A K Peters, 1995.

[3] O. Karch. A Sharper Complexity Bound for the Robot Localization Problem. Technical Report No. 139, Department of Computer Science I, University of Würzburg, June 1996. http://www-info1.informatik.uni-wuerzburg.de/publications/karch/tr139.ps.gz.

[4] K. Mehlhorn and S. Näher. LEDA – A Platform for Combinatorial and Geometric Computing. *Communications of the ACM*, 38:96–102, 1995. http://www.mpi-sb.mpg.de/guide/staff/uhrig/ledapub/reports/leda.ps.Z.

[5] S. Schuierer. Efficient Robot Self-Localization in Simple Polygons, 1996. Unpublished manuscript, accepted for the CG'97 Workshop.