ELSEVIER Discrete Applied Mathematics 93 (1999) 89–108

DISCRETE APPLIED MATHEMATICS

Relocalization – theory and practice

Oliver Karch[a], *, Thomas Wahl[b], 1

[a]Department of Computer Science I, University of Würzburg, Am Hubland, 97074 Würzburg, Germany
[b]ATR Interpreting Telecommunications Research Laboratories, 2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan

Received 27 June 1997; revised 3 March 1998; accepted 22 October 1998

Abstract

We consider the following problem: a robot is at an unknown position in an indoor-environment and has to do a complete relocalization, that is, it has to enumerate all positions that it might be located at. This problem occurs when, for example, the robot "wakes up" after a breakdown (e.g., a power failure or maintenance works) and the possibility exists that it has been moved meanwhile. An idealized version of this problem, where the robot has a range sensor, a polygonal map, and a compass, all of which are exact, that is, without any noise, was solved by Guibas et al. [5]. In the context of their method, we first show that the preprocessing bounds can be expressed slightly sharper. Then we describe an approach to modifying their scheme such that it can be applied to more realistic scenarios (e.g., with uncertain sensors) as well. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Relocalization; Robotics; Sensor uncertainties; Polygon distances

1. The localization problem

We investigate the first stage of the *robot localization problem* [3,11]: an autonomous robot is at an unknown position in an indoor environment, for example a factory building, and has to do a complete relocalization, that is, determine its position and orientation. This is necessary when, for example, the robot "wakes up" after a breakdown (e.g., a power failure or maintenance works) and has no knowledge about its initial configuration or has possibly been moved meanwhile.

For this task, the robot has a polygonal map of its environment and a range sensor (e.g., a laser radar), which provides the robot with an approximation of its visibility polygon. The localization should be performed using only this minimal equipment. In particular, the robot is not allowed to use landmarks (e.g., marks on the walls or on the floor). This should make it possible to use autonomous robots also in fields of application where it is not allowed or too expensive to change the environment.

* Corresponding author. E-mail: karch@informatik.uni-wuerzburg.de.
1 E-mail: twahl@itl.atr.co.jp.

PII: S0166-218X(99)00008-6

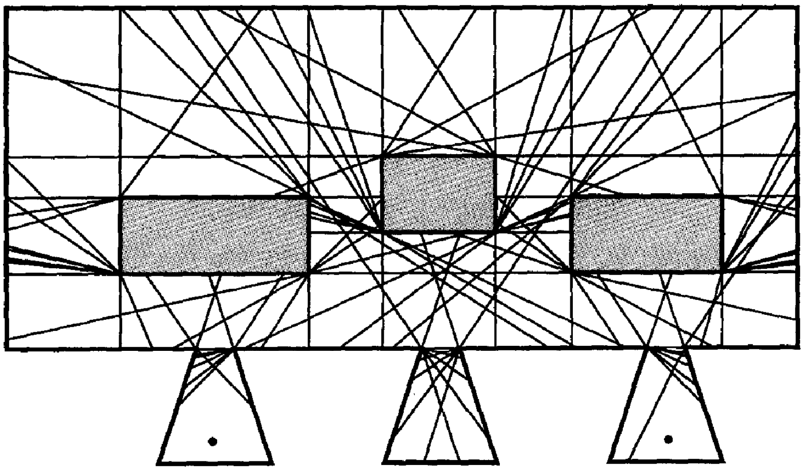


Fig. 1. Polygonal map and its decomposition into visibility cells.

The localization process usually consists of two stages. First, the non-moving robot enumerates all hypothetical positions that are *consistent* with its sensor data, i.e., that yield the same visibility polygon. There can very well be several such positions if the map contains identical parts at different places (e.g., buildings with many identical corridors, like hospitals or libraries). All those positions cannot be distinguished by a non-moving robot. Fig. 1 shows an example: the marked positions at the bottom of the two outermost niches cannot be distinguished using only their visibility polygons.

If there is more than one hypothetical position, the robot eliminates the wrong hypotheses in the second stage and determines exactly where it is by travelling around in its environment. This is a typical on-line problem, because the robot has to consider the new information that arrives while the robot is exploring its environment to find a path as efficient (i.e., short) as possible for eliminating the wrong hypotheses. Dudek et al. [4] have already shown that finding an optimal localization strategy is NP-hard, and described a competitive greedy strategy, the running time of which was recently improved by Schuierer [9].

This paper concentrates on the first stage of the localization process, that is, on generating the possible robot configurations (i.e., positions and orientations). With the additional assumption that the robot already knows its orientation (i.e., the robot has a *compass*) and all sensors and the map are *exact* (i.e., without any noise), this problem turns into a pure geometric one, stated as follows: for a given map polygon $\mathscr{P}$ and a star-shaped polygon $\mathscr{V}$ (the visibility polygon of the robot), find all points $p \in \mathscr{P}$ that have $\mathscr{V}$ as their visibility polygon.

Guibas et al. [5] described a scheme for solving this idealized version of the localization problem efficiently. We will briefly sketch their method in the following section. For some of the occurring complexities we will then give sharper bounds in Section 3.

As this more theoretical method requires exact sensors and an exact map, it is not directly applicable in practice, where the data normally is *noisy*. In Section 4 we consider these problems and show in Sections 5 and 6 an approach to avoiding them, which uses *distance functions* to model the resemblance between the noisy range scans (from the sensor) and the preprocessed skeletons (extracted from the possibly inexact map).

2. Solving the geometric problem

In the following we sketch the method of Guibas et al., for which we will give a sharper preprocessing bound in the next section and which also is the basis for our approach described in Sections 5 and 6. We assume that the robot navigates on a plain surface with mostly vertical walls and obstacles such that the environment can be described by a polygon $\mathcal{P}$, called the *map polygon*. Additionally, we assume that $\mathcal{P}$ has *no holes* (i.e., there are no free-standing obstacles in the environment), although the algorithm remains the same for map polygons with holes; the preprocessing costs, however, may be higher in that case.

The (exact) range sensor generates the star-shaped *visibility polygon* $\mathcal{V}$ of the robot. As the range sensor is only able to measure the distances relative to its own position, we assume the origin of the coordinate system of $\mathcal{V}$ to be the position of the robot.

Using the assumption that we have a compass, the geometric problem is then to find all points $p \in \mathcal{P}$ such that their visibility polygon $\mathcal{V}_p$ is identical with the visibility polygon $\mathcal{V}$ of the robot, that is, the equality $\mathcal{V} + p = \mathcal{V}_p$ holds.

The main idea of Guibas et al. [5] to solving this problem is to divide the map into finitely many visibility cells such that a certain structure (the visibility skeleton, which is closely related to the visibility polygon) does not change inside a cell.

For a localization query we then do not search for points where the visibility polygon fits into the map, but instead for points where the corresponding skeleton does. That is, the continuous problem[2] of fitting a visibility polygon into the map is discretized in a natural way by decomposing the map into visibility cells.

2.1. Decomposing the map into cells

At preprocessing time the map $\mathcal{P}$ is divided into convex *visibility cells* by introducing straight lines forming the boundary of the cells such that the following property holds:

The set of visible map vertices does not change when we travel around within a cell.

As the visibility of a vertex only changes if we cross a straight line induced by that vertex and an occluding *reflex* vertex (i.e., having an internal angle $\geqslant \pi$), the

[2] "Continuous" in the sense that we cannot find an $\varepsilon > 0$ such that the visibility polygon $\mathcal{V}_p$ of a point p moving by at most ε does not change.

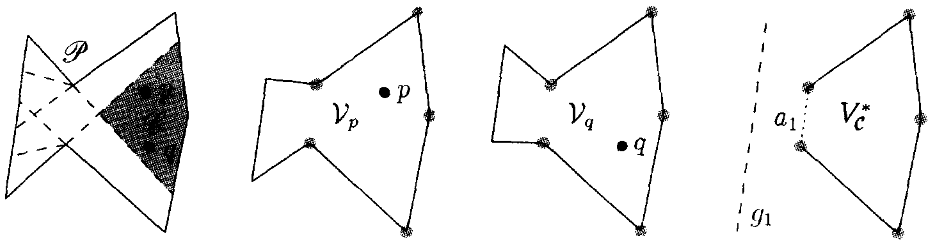


Fig. 2. Decomposition of a map polygon into visibility cells (left), two visibility polygons (middle), and the corresponding skeleton (right).

subdivision into visibility cells can be constructed in the following way: we consider all pairs consisting of a vertex v and a reflex vertex v_r that are visible from each other; for each such pair (v, v_r) we introduce the ray into the map that goes along the line through v and v_r, starts at v_r, and is oriented as to move away from v. An example of such a decomposition is depicted in the left part of Fig. 2. The introduced rays are drawn as dashed lines. The points p and q from cell $\mathscr{C}$ see the same set of five map vertices (marked gray in the corresponding visibility polygons in the middle). Fig. 1 shows a decomposition for a more complex map with three obstacles (gray), generated with our software RoLoPro described in Section 7.

If the map consists of a total number of n vertices, of which r are reflex, the number of introduced rays is in $\mathcal{O}(nr)$ and therefore the complexity of the decomposition is in $\mathcal{O}(n^2r^2)$. For map polygons without holes it can be shown that this complexity is actually in $\mathcal{O}(n^2r)$. Moreover, it is easy to give worst-case examples that show that these bounds are tight.

2.2. The visibility skeleton

When we compare two visibility polygons of points from the same cell (see Fig. 2), we see that they are very similar and differ only in certain aspects, namely in the *spurious edges* that are caused by the reflex vertices and are collinear with the viewpoint, and in those map edges that are only *partially visible.* The remaining *full edges* (which are completely visible) are the same in both polygons. This observation can be used to define a structure that does not change inside a visibility cell, the *visibility skeleton.*

For a visibility polygon $\mathcal{V}_p$ with viewpoint p, the corresponding visibility skeleton V_p^* is constructed by removing the spurious edges, and by substituting the partially visible edges (they lie between two spurious vertices or between a spurious and a full vertex) with an *artificial edge* a_i together with the corresponding line g_i on which the original (partially visible) edge of $\mathcal{V}_p$ lies. Thus, we simply ignore the spurios edges and the spurios vertices, as this information continuously depends on the exact position p.

As the skeleton does not change inside a cell, we can define the *cell skeleton* $V_{\mathscr{C}}^*$ as the common skeleton of all visibility polygons of points from cell $\mathscr{C}$. Fig. 2 shows an example of the common skeleton $V_{\mathscr{C}}^*$ of two visibility polygons $\mathcal{V}_p$ and $\mathcal{V}_q$ for points p and q from the same cell $\mathscr{C}$.

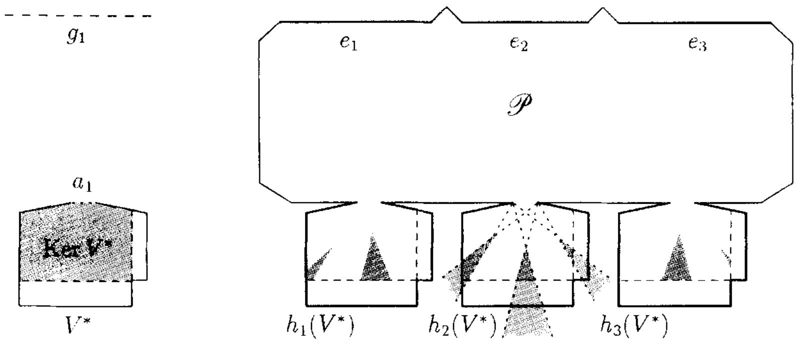


Fig. 3. Three embeddings of the skeleton V^*.

2.3. Embeddings of a skeleton

Just as a star-shaped polygon $\mathscr{V}$ can fit into the map $\mathscr{P}$ at several positions, the same holds for the skeleton V^* defined above. A mapping $h(V^*)$, which fits V^* into the map $\mathscr{P}$, is called an *embedding* of the skeleton V^*. It can be shown that for every skeleton only $\mathcal{O}(r)$ different embeddings exist (r being the number of reflex vertices). Therefore, for a localization query the maximum number of possible robot locations is also bounded by $\mathcal{O}(r)$.

The connection between a skeleton and its embeddings is illustrated in Fig. 3. The skeleton V^*, which has one artificial edge a_1, has three embeddings $h_1,\ldots,h_3$ into the map $\mathscr{P}$. In each embedding, each full edge of V^* matches an edge of $\mathscr{P}$. Furthermore, for the artificial edge a_1 there exists at least one map edge that lies on the corresponding embedded line $h_j(g_1)$, for $1 \leqslant j \leqslant 3$. Such edges are called *candidate edges*.

For a fixed skeleton V^* with an artificial edge a_i, which is embedded into the map by h_j, we denote by $C_{i,j}$ the set of all candidate edges for a_i in embedding $h_j(V^*)$ and by $c_{i,j}$ the cardinality of $C_{i,j}$. Note that one edge may serve as a candidate edge in several embeddings of the same skeleton. For example, in Fig. 3 the sets of candidate edges for the embeddings of the skeleton V^* are $C_{1,1} = C_{1,2} = C_{1,3} = \{e_1, e_2, e_3\}$. The visibility cells with skeleton V^* are drawn in dark-gray in the figure. Note that from each of these cells exactly one of the candidate edges $e_1,\ldots,e_3$ for the (embedded) artificial edge $h_j(a_1)$ is visible "through" a_1.

When we construct the skeleton V_p^* from the visibility polygon $\mathscr{V}_p$, we "throw away" some information about the spurious edges and the partially visible edges. But this information can be reconstructed using the position of the viewpoint p relative to the skeleton. This relationship between a visibility polygon and the embeddings of its corresponding skeleton is described in the following theorem.

Theorem 1 (Guibas et al. [5]). *Let $\mathscr{V}_p$ be a visibility polygon, V_p^* the corresponding skeleton, $h(V_p^*)$ an embedding of this skeleton into the map, and $h(p)$ the corresponding viewpoint in the embedding. Then, $\mathscr{V}_{h(p)} = \mathscr{V}_p$ if and only if $V_{h(p)}^* = V_p^*$.*

The consequence of this theorem is that in order to determine all points in the map that have $\mathscr{V}_p$ as their visibility polygon it is sufficient to consider all embeddings $h(V_p^*)$ of the skeleton V_p^* and then to check whether the corresponding embedded viewpoint $h(p)$ induces the same skeleton V_p^*, that is, the point $h(p)$ must lie in a cell $\mathscr{C}$ with $V_{\mathscr{C}}^* = V_p^*$.

That means, the reduction of the visibility polygon to the corresponding skeleton and the decomposition of the map into visibility cells discretizes our problem in a natural way: instead of testing infinitely many visibility polygons we have to test only a finite number of skeletons to determine all possible robot locations.

2.4. *Costs of the localization query and the preprocessing*

As already stated above, the decomposition into visibility cells has a complexity of $\mathcal{O}(n^2r)$ for map polygons without holes. At preprocessing time this decomposition is computed and the cells are divided into equivalence classes according to their skeletons, that is, two cells $\mathscr{C}_1$ and $\mathscr{C}_2$ are said to be *equivalent* if $V_{\mathscr{C}_1}^*$ equals $V_{\mathscr{C}_2}^*$ up to a translation. For the resulting set of equivalence classes a search structure (e.g., a multidimensional search tree) is constructed that allows for a given skeleton the retrieval of the corresponding class in an efficient way (i.e., in time logarithmic in the number of classes).

For the localization query, the skeleton V_p^* of the given visibility polygon $\mathscr{V}_p$ is computed and the corresponding equivalence class is determined. As we know the position of the point p relative to the skeleton V_p^* and as we also know the position of each cell $\mathscr{C}$ relative to its cell skeleton $V_{\mathscr{C}}^*$, we can easily determine for each cell in the equivalence class the embedded point $h(p)$ and check whether it lies in $\mathscr{C}$. If the point $h(p)$ is in $\mathscr{C}$, then $h(p)$ is a valid robot position by Theorem 1. The test whether $h(p)$ is in $\mathscr{C}$ or not can be performed very efficiently by using a point location structure; in fact, only a single point location query is necessary.

This way we get a query time of $\mathcal{O}(m+\log n+A)$ where m is the number of vertices of $\mathscr{V}_p$ and A denotes the size of the output, that is, the number of all reported robot locations.

The total preprocessing time and space is in $\mathcal{O}(n^2r \cdot |\mathscr{E}\mathscr{C}|)$ for map polygons without holes. Here, $|\mathscr{E}\mathscr{C}|$ denotes the worst-case complexity of an equivalence class, where for a given skeleton V^* the complexity of the corresponding equivalence class $\mathscr{E}\mathscr{C}_{V^*}$ is the total number of vertices and edges of all visibility cells with skeleton V^*. In [5] it was already shown that $|\mathscr{E}\mathscr{C}|$ is in $\mathcal{O}(n^2)$.

3. A sharper bound for $|\mathscr{E}\mathscr{C}|$

In this section we establish a tighter upper bound of $\mathcal{O}(n + r^2)$ for the complexity of an equivalence class, such that the dependence on the number of reflex vertices becomes clearer; we also show that this bound is worst-case optimal. Due to lack of space the proofs will only be sketched. The complete proofs are given in [6].

The main idea of the proof is to concentrate on a single skeleton V^* and to determine the complexity of $\mathscr{E}\mathscr{C}_{V^*}$ by counting the edges of all visibility cells whose skeleton equals V^*. Therefore, we first study the structure of the visibility cells and classify their bounding edges into two groups. It turns out that only the edges of the second type, which are determined by the candidate edges, are difficult to count, and we will first give an upper bound for their number in a *single* embedding of V^*.

Unfortunately, summing up these numbers for all embeddings of the skeleton V^* does not yield the desired bound. The reason is that one edge may serve as a candidate edge in several embeddings as already stated above. Therefore, we will examine such situations and show how to perform the summation in a more sophisticated way.

3.1. The structure of the visibility cells

For a fixed skeleton V^*, we examine the visibility cells whose visibility skeleton equals V^*. Each edge of such a visibility cell is either a *kernel edge* or an *AC edge*:

Kernel edges are edges that lie on the boundary of the embedded *kernel* $h(\operatorname{Ker} V^*)$ of the skeleton. Here, the kernel $\operatorname{Ker} V^*$ of a skeleton V^* is defined analogously to the kernel of a polygon (see the left part of Fig. 3): $\operatorname{Ker} V^*$ consists of all points that are visible from all edges of the skeleton (i.e., from all points of all full edges and all artificial edges). For example, all horizontal and vertical edges of the visibility cells of Fig. 3 are kernel edges.

AC edges are determined by pairs consisting of an embedded artificial edge and a corresponding candidate edge. An AC edge lies on the ray induced by a reflex vertex of the artificial edge and a vertex of the candidate edge. In Fig. 3 these rays are drawn as dotted lines starting at the vertices of the embedded artificial edge $h_2(a_1)$.

The next lemma, the proof of which we omit, gives us an upper bound for the total number of kernel edges. It shows that it suffices to count only the AC edges for establishing the $\mathcal{O}(n+r^2)$ bound.

Lemma 2. *The total number of kernel edges in $\mathscr{E}\mathscr{C}_{V^*}$ is in $\mathcal{O}(n+r^2+t)$, where t is the total number of AC edges.*

In order to count all AC edges we consider each embedding h_j of the skeleton V^* and count the number of AC edges in $h_j(V^*)$. Summing up over all embeddings yields the total number of AC edges in $\mathscr{E}\mathscr{C}_{V^*}$. To this end, let k be the number of artificial edges of V^* and let s be the number of all embeddings of V^*. Recall that s (as well as k) is in $\mathcal{O}(r)$, the number of reflex vertices.

The following lemma bounds the number of AC edges in a single embedding $h_j(V^*)$.

Lemma 3. *The number of all AC edges in embedding $h_j(V^*)$ is in*

$$\mathcal{O}\left(\sum_{i=1}^{k} c_{i,j} + \left(\sum_{i=1}^{k} c_{i,j}\right)^2 - \sum_{i=1}^{k} c_{i,j}^2\right). \tag{1}$$

Proof. This can be shown by considering how the visibility cells are created: each pair consisting of an embedded artificial edge and a candidate edge induces a *visibility wedge*. This wedge consists of all points that can see the candidate edge "through" the embedded artificial edge. Fig. 3 shows the visibility wedges (drawn in light-gray) of $h_2(V^*)$ as an example.

It can easily be seen that each AC edge lies on the boundary of one visibility wedge. Therefore, the complexity of the arrangement of the visibility wedges for all artificial and corresponding candidate edges of $h_j(V^*)$ gives us an upper bound for the number of AC edges. When we take into account that each visibility wedge in $h_j(V^*)$ corresponds to a candidate edge in one of the sets $C_{i,j}$, for $1 \leqslant i \leqslant k$, this complexity is contained in the class stated in (1). □

3.2. The total number of possible candidate edges

Since the number of candidate edges plays an important role in Lemma 3, we establish an upper bound for the total number of all *possible* candidate edges of cells in $\mathcal{EC}_{V^*}$. To this end, let $\overline{C}$ be the set of all candidate edges for all artificial edges in all embeddings of the skeleton V^*, that is

$$\overline{C} := \bigcup_{\substack{1 \leqslant i \leqslant k \\ 1 \leqslant j \leqslant s}} C_{i,j}.$$

For the cardinality of $\overline{C}$ the following can be shown:

Lemma 4. *In map polygons without holes or with only convex holes, the cardinality of $\overline{C}$ is in $\mathcal{O}(r)$.*

Idea of Proof. Basically, this holds because at least one reflex vertex lies between two consecutive candidate edges. For example, in Fig. 3 there are two reflex vertices between edges e_1 and e_2. □

If we assume that the sets $C_{i,j}$ of candidate edges for a *single* embedding $h_j(V^*)$ are disjoint,[3] the sum $\sum_{i=1}^{k} c_{i,j}$ is in $\mathcal{O}(r)$ by Lemma 4 and using Lemma 3 we obviously get an upper bound of $\mathcal{O}(r^2)$ for the number of AC edges in a *single* embedding.

Furthermore, if we could expect that *all* sets $C_{i,j}$ of candidate edges are disjoint, each possible candidate edge of $\overline{C}$ could be assigned to *exactly one* visibility wedge and the complexity of the arrangement of the $\mathcal{O}(|\overline{C}|)$ visibility wedges in *all* embeddings

[3] Note that this assumption does *not hold* if the map polygon is allowed to have holes.

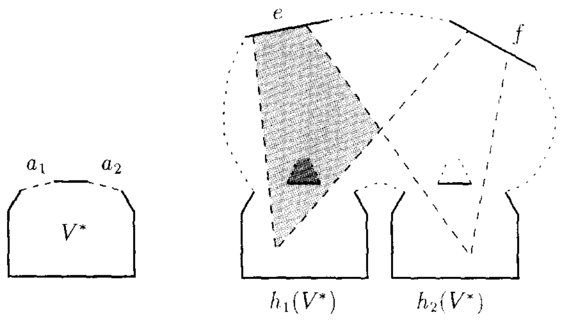


Fig. 4. Assigning two candidate edges to two pairs of artificial edges.

of V^* would also be bounded by $\mathcal{O}(|\overline{C}|^2) \subseteq \mathcal{O}(r^2)$. That is, we would have established our desired bound of $\mathcal{O}(r^2)$ for the total number of AC edges. But unfortunately, this assumption is not true, since the sets $C_{i,j}$ need not be disjoint as already noted in the example in Section 2.3.

Therefore, we have to take a closer look at situations where one edge may serve as a candidate edge in more than one embedding of V^*. In this context it is useful to take *pairs* of candidate edges into account.

3.3. Pairs of possible candidate edges

Although one possible candidate edge of $\overline{C}$ may be contained in more than one of the sets $C_{i,j}$, the following lemma shows that for one *pair* $(e,f) \in \overline{C} \times \overline{C}$ of possible candidate edges there exist at most one embedding h_j and at most one pair of artificial edges such that e and f are candidate edges for the two artificial edges in embedding $h_j(V^*)$. Informally speaking, each pair of candidate edges can be assigned to at most one pair of visibility wedges.

Lemma 5. *In map polygons without holes, the cardinality of the set*

$$\{(i_1,i_2,j) \mid i_1 \neq i_2 \wedge (e,f) \in C_{i_1,j} \times C_{i_2,j}\}$$

is at most one, for every pair $(e,f) \in \overline{C} \times \overline{C}$.

Proof. Omitting the details and simplifying the situation, the argument is as follows: assume that for a skeleton V^* two pairs of visibility wedges exist that are induced by the same pair (e,f) of possible candidate edges. This situation is depicted in Fig. 4: the skeleton V^* has two embeddings h_1 and h_2 such that the edge e determines a visibility wedge for the artificial edges $h_1(a_1)$ and $h_2(a_1)$. Analogously, f determines a wedge for $h_1(a_2)$ and $h_2(a_2)$. (Note that the lines g_1 and g_2 and the visibility wedges are omitted in the figure.) As e and f must be visible "through" the embedded artificial edges, there must exist lines of sight (drawn dashed in the figure) from the candidate edges e and f, respectively, to points inside the embeddings $h_1(V^*)$ and $h_2(V^*)$, respectively. These lines intersect and create a circle (drawn in light-gray) that does not intersect any map edge and contains at least one map vertex. Therefore, the map must have at least one hole (drawn in dark-gray), which contradicts our assumption that the map has no holes. □

Using the same idea as in this proof, it can furthermore be shown that for each possible candidate edge e and for each embedding h_j, at most one artificial edge exists such that e is a candidate edge in embedding $h_j(V^*)$, which corresponds to the artificial edge. Informally speaking, in each embedding each candidate edge can be assigned to at most one visibility wedge. This fact is expressed in the following lemma.

Lemma 6. *In map polygons without holes, the cardinality of the set* $\{i \mid e \in C_{i,j}\}$ *is at most one for every* $e \in \overline{C}$ *and for* $1 \leqslant j \leqslant s$.

Now, we are able to show the $\mathcal{O}(r^2)$ bound for the total number of AC edges. Using (1) from Lemma 3, this number is in

$$\mathcal{O}\left(\sum_{j=1}^{s}\sum_{i=1}^{k} c_{i,j} + \sum_{j=1}^{s}\left(\left(\sum_{i=1}^{k} c_{i,j}\right)^2 - \sum_{i=1}^{k} c_{i,j}^2\right)\right).$$

This sum can be rewritten as

$$\sum_{j=1}^{s}\sum_{i=1}^{k} |C_{i,j}| + 2\sum_{j=1}^{s}\sum_{\substack{1\leqslant i_1,i_2\leqslant k\\ i_1<i_2}} |C_{i_1,j}\times C_{i_2,j}|,$$

and applying Lemmas 5 and 6 yields

$$\sum_{j=1}^{s}|\overline{C}| + 2\left|\bigcup_{j=1}^{s}\bigcup_{\substack{1\leqslant i_1,i_2\leqslant k\\ i_1<i_2}} C_{i_1,j}\times C_{i_2,j}\right|.$$

Using Lemma 4 and the fact that $s \in \mathcal{O}(r)$, this number is in $\mathcal{O}(r|\overline{C}| + |\overline{C}|^2) \subseteq \mathcal{O}(r^2)$.

Summarizing our results, we get the following theorem.

Theorem 7. *The total complexity of any equivalence class for a map polygon without holes is in* $\mathcal{O}(n + r^2)$.

In the case of many reflex vertices (i.e., $r \in \Omega(n)$) this yields the same bound as [5]. However, the dependence on the number of reflex vertices is now expressed more clearly. For example, if the number of reflex vertices r is bounded by $\mathcal{O}(\sqrt{n})$, the complexity of an equivalence class depends only linearly and not quadratically on the total number n of map vertices.

3.4. Effects on the preprocessing costs

Using Theorem 7 and the same arguments as in [5], the time and space bounds for the preprocessing (see Section 2) can be sharpened from $\mathcal{O}(n^4 r)$ to $\mathcal{O}(n^2 r \cdot (n + r^2))$.

3.5. *A worst-case example*

Consider the map polygon shown in Fig. 3, which can be looked at as a corridor with large niches on one side and small niches on the other side. If we insert additional collinear candidate edges e_i and additional large niches $h_j(V^*)$, we get a map polygon with $\Omega(r)$ embeddings of V^*, each with $\Omega(r)$ possible candidate edges for the artificial edge a_1 in embedding $h_j(V^*)$. If we scale the scene in an appropriate way, we can achieve that each edge e_i induces a visibility cell with skeleton V^* in each embedding $h_j(V^*)$. Therefore, the total number of AC edges of all visibility cells equivalent to V^* is in $\Omega(r^2)$. Furthermore, by appropriately inserting additional points to V^* we can achieve that the number of kernel edges is in $\Omega(n)$.

Therefore, we get a total complexity of $\Omega(n + r^2)$ for the equivalence class of the skeleton V^* for this worst-case example.

3.6. *Map polygons with holes*

The above results only hold for map polygons *without holes*. That is, no free-standing obstacles are allowed in the robot's environment. For map polygons with holes, the bounds become worse (see [5,6]). But for the special case of map polygons with l *convex holes*, an upper bound of $\mathcal{O}(n + (l + 1)r^2)$ for the worst-case complexity of an equivalence class can be proven.

The idea for establishing this bound is essentially the same as in the case without holes: as in that proof, it can be shown that the cardinality of the two sets defined in Lemmas 5 and 6 is at most l (instead of one). Again, this is done by introducing lines of sight from points inside the embedded skeletons to the candidate edges and counting the number of created holes.

4. Problems in realistic scenarios

The idealizing assumptions of the method described in Section 2 prevent us from using it in realistic scenarios, as we encounter several problems:

- Realistic range sensors do not generate a visibility *polygon* $\mathcal{V}$ as assumed for the method, but only a finite sequence $\mathcal{S}$ of *scan points* (usually, measured at equidistant angles). Furthermore, these scan points do not lie exactly on the robot's visibility polygon, but are perturbed due to sensor uncertainties. An example is depicted in Fig. 5, which shows the exact visibility polygon and a (simulated) noisy scan for a robot standing in the left niche of Fig. 1. Even if we connect the scan points by straight-line segments, we only get an approximation $\mathcal{V}_{\mathcal{S}}$ of the robot's exact visibility polygon $\mathcal{V}$.
- For the localization process we assume that we already know the exact orientation of the robot. But in practice this is often not the case, and we only have inexact knowledge or no knowledge at all about the robot's orientation.

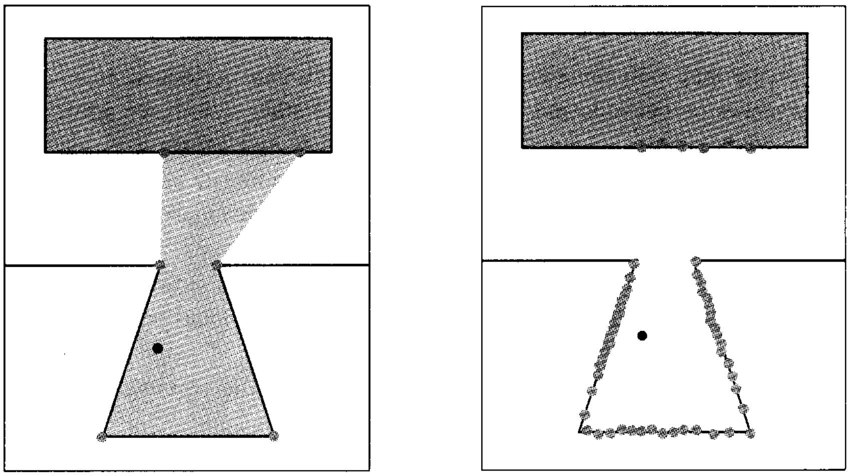


Fig. 5. Exact visibility polygon (left) and noisy scan (right).

- There may be obstacles in the environment that are not considered in the map and which may affect the robot's view. For example, furniture that is too small to be considered for map generation or even dynamic obstacles like people or other robots.
- Realistic range sensors have a limited sensing range and obstacles that have a greater distance to the robot cannot be detected.

The consequence is that the (approximated) visibility skeleton $V^*_{\mathscr{S}}$, which the robot computes from its approximated visibility polygon $\mathscr{V}_{\mathscr{S}}$, usually does not match any of the preprocessed skeletons exactly. That is, the robot is not able to determine the correct equivalence class, and the localization process completely fails.

5. Adaptation to practice

Our approach to tackling these problems is, for a given range scan $\mathscr{S}$ (from the sensor), to search for the preprocessed skeleton that is *most similar* to the scan. For modeling the resemblance between a scan $\mathscr{S}$ and a skeleton V^* we use an appropriate *distance function* $d(\mathscr{S}, V^*)$. Then, instead of performing an *exact match* query as in the original algorithm, we carry out a *nearest-neighbor* query in the set of skeletons with respect to the chosen distance function $d(\mathscr{S}, V^*)$ to find the skeleton with the highest resemblance to the scan.

Depending on the distance function, we then additionally have to apply a matching algorithm to the scan and the skeleton in order to determine the position of the robot. The reason is that not all methods for determining a distance measure yield an optimal matching (i.e., a translation vector and a rotation angle) as well. Consider, for example, the algorithm for computing the *Arkin metric* [2] for polygons, which, besides the distance measure, only provides the optimal rotation angle and no translation vector. In contrast to this, algorithms for computing the *minimum Hausdorff distance* (under rigid motions) [1] provide both, the distance measure and the corresponding matching.

5.1. Requirements to the distance function

In order to be useful in practice, a distance function $d(\mathscr{S}, V^*)$ should at least have the following properties:

Continuity. The distance function should be continuous in the sense that small changes in the scan (e.g., caused by noisy sensors) or even in the skeleton (e.g., caused by an inexact map) should only result in small changes of the distance. More precisely: Let $d_{\mathscr{S}}(\mathscr{S}_1, \mathscr{S}_2)$ and $d_{V^*}(V_1^*, V_2^*)$ be functions that measure the resemblance between two scans $\mathscr{S}_1$ and $\mathscr{S}_2$ and between two skeletons V_1^* and V_2^*, respectively. An appropriate reference distance measure for $d_{\mathscr{S}}(\mathscr{S}_1, \mathscr{S}_2)$ and $d_{V^*}(V_1^*, V_2^*)$ is, for example, the Hausdorff distance (see Section 6.1).

The distance $d(\mathscr{S}, V^*)$ is said to be *continuous with respect to scans* if

$$\forall_{\varepsilon>0}\exists_{\delta>0} : d_{\mathscr{S}}(\mathscr{S}_1, \mathscr{S}_2) < \delta \Rightarrow |d(\mathscr{S}_1, V^*) - d(\mathscr{S}_2, V^*)| < \varepsilon$$

holds, for all scans $\mathscr{S}_1, \mathscr{S}_2$ and all skeletons V^*. Analogously, $d(\mathscr{S}, V^*)$ is said to be *continuous with respect to skeletons* if

$$\forall_{\varepsilon>0}\exists_{\delta>0} : d_{V^*}(V_1^*, V_2^*) < \delta \Rightarrow |d(\mathscr{S}, V_1^*) - d(\mathscr{S}, V_2^*)| < \varepsilon$$

holds, for all skeletons V_1^*, V_2^* and all scans $\mathscr{S}$.

The requirement of continuity is also motivated by the fact that particularly the classification of the edges of the visibility polygon into different types (spurious edges, partially visible edges, etc.) makes the original method susceptible to perturbations: even a small translation of a vertex can change the type of an edge which yields a skeleton that does not match any equivalence class. In this sense, the exact match query of the original method can also be interpreted as a discrete distance between a visibility polygon and a skeleton, which, however, strongly violates the continuity requirement, because it takes only two values (e.g., 0 – “match” and 1 – “no match”).

Similarity preservation. A skeleton V^* that is similar to $\mathscr{S}$ should have a small distance value $d(\mathscr{S}, V^*)$. Otherwise, the distance would not give any advice for finding a well-matching skeleton and therefore be useless for the localization algorithm. In particular, if we take a scan $\mathscr{S}$ from a point p whose skeleton equals V^*, we want the distance $d(\mathscr{S}, V^*)$ to be zero or at least small, depending on the amount of noise and the resolution of the scan.

Translational invariance. As the robot has no knowledge about the relative position of the coordinate systems of the scan and the skeleton to each other, a translation of the scan or the skeleton in their local coordinate systems must not influence the distance. Rather finding this position is the goal of the localization algorithm.

Rotational invariance. If the robot does not have a compass, the distance must also be invariant under rotations of the scan (or the skeleton, respectively).

Fast computability. As the distance $d(\mathcal{S}, V^*)$ has to be determined several times for a single localization query (for different skeletons, see Section 5.2), the computation costs should not be too high.

As we do not want to compare a scan with all skeletons to find the skeleton with the highest resemblance (remember that their number can be in $\Omega(n^2r^2)$, see Section 2.1), the skeletons should be stored in an appropriate data structure that we can search through efficiently.

5.2. Maintaining the skeletons

For this purpose we can use the *monotonous bisector tree* [8], a spatial index that allows to partition the set of skeletons hierarchically with respect to a second distance function $D(V_1^*, V_2^*)$ that models the resemblance between two skeletons V_1^* and V_2^*. The set of skeletons is recursively divided into clusters with monotonously decreasing cluster radii in the preprocessing step. This division then represents the similarities of the skeletons among each other.

The distance function $D(V_1^*, V_2^*)$ should be chosen "compatible" to the function $d(\mathcal{S}, V^*)$, such that in the nearest-neighbor query not all clusters have to be investigated. That is, at least the *triangle inequality*

$$d(\mathcal{S}, V_2^*) \leqslant d(\mathcal{S}, V_1^*) + D(V_1^*, V_2^*)$$

should be satisfied. This way, we can determine lower bounds for the distance values $d(\mathcal{S}, V^*)$ of complete clusters, when traversing the tree. Such a cluster can then be rejected and does not have to be examined.

6. Suitable distances for $d(\mathcal{S}, V^*)$

It is hard to find distance functions that have all the properties from Section 5.1. Particularly, the fifth requirement is contrary to the remaining ones. Moreover, it is often not possible to simply use existing polygon distances, because in our problem we have to cope with scans and skeletons instead of polygons. Therefore, a careful adaptation of the distance functions is almost always necessary. In the following we investigate two distance functions, the Hausdorff distance and the polar coordinate metric, and illustrate the occurring problems.

6.1. The Hausdorff distance

For two point sets $A, B \subset \mathbb{R}^2$, their *Hausdorff distance* $\delta(A, B)$ is defined as

$$\delta(A,B) := \max\{\vec{\delta}(A,B), \vec{\delta}(B,A)\},$$

where

$$\vec{\delta}(A,B) := \sup_{a \in A} \inf_{b \in B} ||a - b||$$

is the *directed Hausdorff distance* from A to B, and $\|\cdot\|$ is the Euclidean norm. Accordingly, the term $\vec{\delta}(A,B)$ stands for the maximum distance of a point from A to the set B.

Let $\mathscr{T}$ be the set of all Euclidean transformations (i.e., combinations of translations and rotations). The undirected and directed *minimum Hausdorff distances* with respect to these transformations are defined as

$$\delta_{\min}(A,B) := \inf_{t\in\mathscr{T}} \delta(A,t(B)) \quad \text{and} \quad \vec{\delta}_{\min}(A,B) := \inf_{t\in\mathscr{T}} \vec{\delta}(A,t(B)).$$

It can easily be shown that the minimum Hausdorff distances are continuous and by definition also fulfill the third and fourth property of Section 5.1. But their computation is very expensive. According to [1], this can be done in time $\mathcal{O}((ms)^4(m+s)\log(m+s))$ if m is the complexity of the scan and s is the complexity of the skeleton. This is surely too expensive in order to be used in our application.

On the other hand, the computation of the Hausdorff distance without minimization over transformation application is relatively cheap [1], namely in $\mathcal{O}((m+s)\log(m+s))$. The property of continuity is also not affected, but we now have to choose a suitable translation vector and a rotation angle by hand.

An obvious choice for such a translation vector for a scan $\mathscr{S}$ and a skeleton V^* is the vector that moves the scan origin (i.e., the position of the robot) somewhere into the corresponding visibility cell $\mathscr{C}_{V^*}$ (e.g., the center of gravity of $\mathscr{C}_{V^*}$). This is reasonable, because by the definition of the visibility cells, exactly the points in $\mathscr{C}_{V^*}$ induce the skeleton V^*. Of course, the consequence of doing so is that all cells with the same skeleton (e.g., the big cells in the two outermost niches in Fig. 1) must be handled separately, because the distance $d(\mathscr{S},V^*)$ now does not only depend on V^*, but also on the visibility cell itself.[4] Besides, their intersection may be empty and we might not find a common translation vector for all cells. Of course, the bigger the cell is that the scan has to be placed into, the bigger is the error of this approach, compared with the minimum Hausdorff distance.

A compromise for computing a good matching, which does have the advantages of the previous algorithms, is using an *approximate matching strategy*, which yields only a pseudo-optimal solution. This means, the algorithm finds a transformation $t\in\mathscr{T}$ with $\delta(A,t(B))\leqslant c\cdot\delta_{\min}(A,B)$, for a constant $c\geqslant 1$. Alt et al. [1] showed that for any constant $c>1$ an approximate matching with respect to Euclidean transformations can be computed in time $\mathcal{O}(ms\log(ms)\log^*(ms))$ using so-called *reference points*. If we only want an approximate matching with respect to translations instead of Euclidean transformations, the time complexity would even be in $\mathcal{O}((m+s)\log(m+s))$.

Another point to consider is that a skeleton (interpreted as a point set) in general is not bounded, because it includes a straight line for each artificial edge. The result is that the directed distances $\vec{\delta}(V^*,\mathscr{S})$ and $\vec{\delta}_{\min}(V^*,\mathscr{S})$ almost always return an infinite

[4] In this case, the notation $d(\mathscr{S},V^*)$ is a bit misleading, since there might exist several cells that have the same skeleton V^*. To be correct, we should use the notation $d(\mathscr{S},\mathscr{C}_{V^*})$, where the dependence of the distance from the cell is expressed more clearly. But we will use the easier-to-understand expression $d(\mathscr{S},V^*)$.

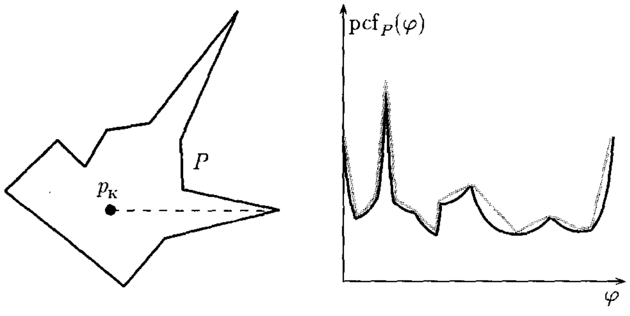


Fig. 6. The function $\mathrm{pcf}_P(\varphi)$ and its linear approximation for the polygon P.

value (except for the trivial case when V^* equals the convex map polygon and has no artificial edges). Therefore, we must either modify the skeletons or we can only use the directed distances $\vec{\delta}(\mathscr{S}, V^*)$ and $\vec{\delta}_{\min}(\mathscr{S}, V^*)$. Note that if we pursue the second approach, the distance $\vec{\delta}_{\min}(\mathscr{S}, V^*)$ is also similarity preserving, provided that the resolution of the scan is high enough such that no edge, in particular, no artificial edge, is missed.

6.2. The polar coordinate metric

A more specialized distance for our problem than the Hausdorff distance is the *polar coordinate metric* (PCM for short) investigated by Wahl [10], which takes a fundamental property of our problem into account: all occurring polygons are *star-shaped* in the following sense, and we even know a kernel point:

- The approximate visibility polygon $\mathscr{V}_{\mathscr{S}}$ (generated from the scan points) is star-shaped by construction with the origin as a kernel point.
- Every skeleton V^* is star-shaped in the sense that from every point in the corresponding visibility cell $\mathscr{C}_{V^*}$ all full edges are completely visible, and for each artificial edge a_i a part of the corresponding straight line g_i is visible (cf. the definition of the kernel of a skeleton in Section 3.1).

To compute the PCM between two (star-shaped) polygons P and Q with kernel points p_K and q_K we first define the value of the *polar coordinate function* (PCF):

$$\mathrm{pcf}_P(\varphi) : \mathbb{R} \to \mathbb{R}_{\geqslant 0}$$

as the distance from the kernel point p_K to the intersection point of a ray starting at p_K in direction φ with the boundary of P. That is, the function $\mathrm{pcf}_P(\varphi)$ corresponds to a description of the polygon P in polar coordinates (with p_K as the origin) and is periodical with a period of 2π. Fig. 6 depicts the PCF (drawn in black) for a star-shaped polygon as an example. In the same way we define the function $\mathrm{pcf}_Q(\varphi)$ for the polygon Q.

Then, the PCM between the polygons P and Q is the minimum integral norm between the functions pcf_P and pcf_Q in the interval $[0, 2\pi]$ over all horizontal translations between the two graphs (i.e., rotations between the corresponding polygons):

$$\mathrm{pcm}(P,Q) := \min_{t\in[0,2\pi]} \sqrt{\int_0^{2\pi} (\mathrm{pcf}_P(\varphi - t) - \mathrm{pcf}_Q(\varphi))^2 \mathrm{d}\varphi}. \tag{2}$$

For a fixed kernel point the function pcf_P is continuous in φ except for one special case: when we move a vertex of a polygon edge such that the edge becomes collinear to p_K, the function pcf_P has a discontinuity at the corresponding angle, the height of which represents the length of the collinear edge. Moreover, the PCF is also continuous in the sense of the definitions in Section 5.1 with respect to translations of the polygon vertices or translations of the kernel point unless this special case occurs. But as pcf_P and pcf_Q may have only finitely many such discontinuities, the integration makes them continuous with respect to *all* translations of polygon vertices and translations of the kernel points, provided that P and Q remain star-shaped with kernel points p_K and q_K.

It can easily be seen that the PCM fulfils the continuity requirement of Section 5.1, if the kernel points are considered as a part of the polygons (i.e., part of the *input* of the PCM). This means that, given two polygons P and Q and an $\varepsilon > 0$, we can find a $\delta > 0$ such that $|\mathrm{pcm}(P,Q) - \mathrm{pcm}(P',Q)| < \varepsilon$, for all polygons P' that are created from P by moving all vertices *and* the kernel point p_K by at most δ. Moreover, if the kernel points are *not* considered as input of the PCM (that is, they are computed from P and Q by the algorithm that computes $\mathrm{pcm}(P,Q)$), the PCM is continuous as well, provided that the kernel points depend continuously on the polygons. For example, the center of gravity of the kernel of a polygon P depends continuously on P and can be used as a kernel point p_K, whereas the left-most kernel point does *not* depend continuously on the polygon.

Wahl [10] showed that the function $\mathrm{pcm}(P,Q)$ is a polygon *metric*, provided that the kernel points are invariant under Euclidean transformations. That is, if p'_K denotes the kernel point of a polygon $P' = t(P)$ for a transformations $t \in \mathscr{T}$, the equality $t(p_K) = p'_K$ must hold, for all polygons P and all $t \in \mathscr{T}$. For example, the center of gravity of the kernel of the polygon has this property.

Furthermore, a linear approximation of the PCM is introduced, which also has all metric properties and is sufficient for our applications. This approximation is depicted in Fig. 6: the points corresponding to a polygon vertex and the local minima of the PCF are connected by straight-line segments (drawn in gray) to get a modified PCF. The minimum integral norm is then defined like in the non-approximated version of the PCM (see Eq. (2)). Following an idea of Arkin et al. [2], the actual computation of the minimum integral norm between the two *piecewise linear* functions can now be carried out much faster than the computation of the original PCM: Arkin et al. compute the minimum integral norm between two *piecewise constant* functions in time $\mathcal{O}(pq)$. This idea can be generalized to compute the approximated PCM in time $\mathcal{O}(pq\cdot(p+q))$, where p and q stand for the complexities of the two polygons. Of course, if we do not want to minimize over the rotations, the computation time is only in $\mathcal{O}(p+q)$, even for the non-approximated version.

If we want to use the PCM as a distance function $d(\mathscr{S}, V^*)$ we need corresponding star-shaped polygons for $\mathscr{S}$ and V^*:

- For the scan, we choose the approximated visibility polygon $\mathscr{V}_{\mathscr{S}}$, which is star-shaped by construction. Again, the coordinate origin can be used as a kernel point.
- For generating a polygon from a skeleton V^* (with corresponding cell $\mathscr{C}_{V^*}$), we choose a point c inside the cell $\mathscr{C}_{V^*}$ (e.g., the center of gravity) and determine the visibility polygon $\mathscr{V}_c$ of this point. By construction, c is a kernel point of $V^*_{\mathscr{C}}$.

With this choice we yield the following theorem about the polar coordinate metric as a distance function.

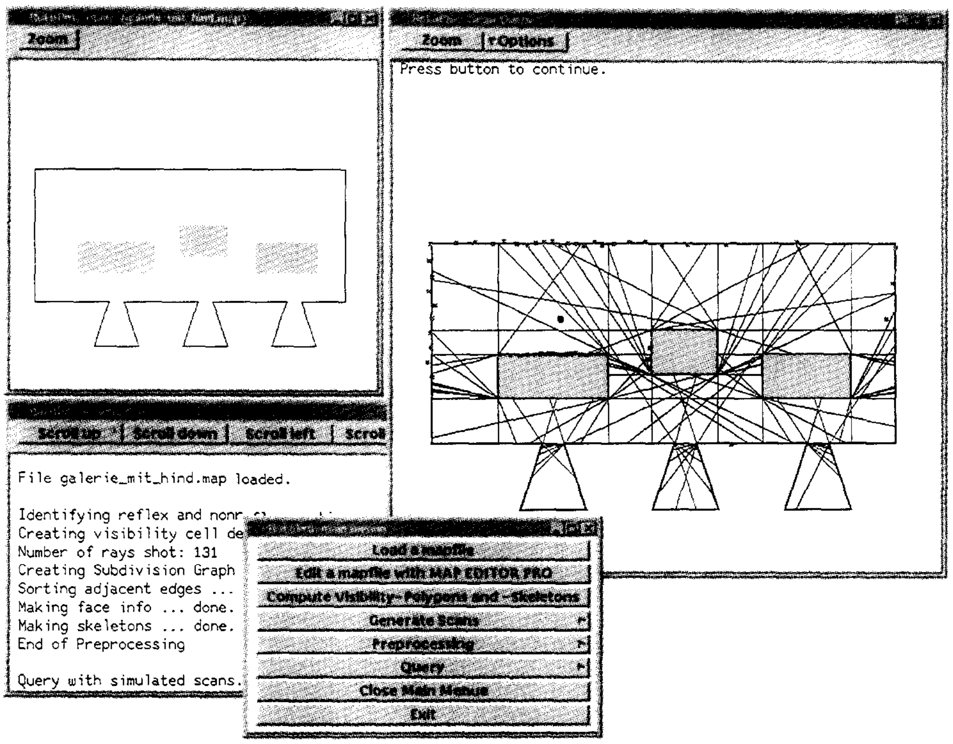
Theorem 8. *The distance function $d(\mathscr{S}, V^*) := \mathrm{pcm}(\mathscr{V}_{\mathscr{S}}, \mathscr{V}_c)$, with $\mathscr{V}_{\mathscr{S}}$ and $\mathscr{V}_c$ as defined above, fulfills the following requirements from Section 5.1: continuity, invariance against translations and rotations, and fast computability.*

Note that the PCM is not similarity preserving: if the point c chosen above for computing a corresponding polygon for a visibility cell $\mathscr{C}$ does not equal the robot's position, the two polygons that are compared by the PCM are different and their distance value cannot be zero. But in practice, the visibility cells usually are not too large. That means, if we take a scan at a position $p \in \mathscr{C}$, the distance from p to the corresponding point $c \in \mathscr{C}$ is not too large. Thus, the approximated visibility polygon $\mathscr{V}_{\mathscr{S}}$ and the visibility polygon $\mathscr{V}_c$ differ not too much, and the value of $\mathrm{pcm}(\mathscr{V}_{\mathscr{S}}, \mathscr{V}_c)$ is small.

7. Implementation and first experimental results

We have implemented the two versions of the localization algorithm in C++ using the LEDA Library of Efficient Datatypes and Algorithms [7], namely the original method described in Section 2 for exact sensors as well as the modification for realistic scenarios introduced above. Here, the original algorithm was modified and simplified at some points, since we did not focus our efforts on handling sophisticated but rather complicated data structures and algorithmic ideas that were suggested by Guibas et al. Rather, we wanted to have an instrument to experiment with different inputs for the algorithm that is reasonably stable to be used in real-life environments in the future and that can serve as a basis for own modifications. A consequence is that the program does not keep to all theoretical time and space bounds proven in [5,6], as this would have required a tremendous programming effort. Nevertheless, it is reasonably efficient. Fig. 7 shows a screen shot of our robot localization program RoLoPro, which is processing a localization query for a (simulated) noisy scan.

As distance function $d(\mathscr{S}, V^*)$ we have implemented the Hausdorff distance and the polar coordinate metric described in Section 6. First tests in small scenes have shown a success rate of approximately 60% for the Hausdorff distance, i.e., the scan origins of



```
Press button to continue.
```

```
File galerie_mit_hind.map loaded.

Identifying reflex and nonr
Creating visibility cell de
Number of rays shot: 131
Creating Subdivision Graph
Sorting adjacent edges ...
Making face info ... done.
Making skeletons ... done.
End of Preprocessing

Query with simulated scans.
```

Fig. 7. Screen shot of RoLoPro.

about 60 out of 100 scans were inside that cell with the smallest distance to the scan. In the same scenes the success rate of the polar coordinate metric was about 90%.

8. Future work

Currently we are testing our approach described in the last sections with our simulation software and we are going to evaluate it in two different scenarios for service robots.

Our main goals for the future are:

- We are going to implement the efficient skeleton management described in Section 5.2 to improve the running time of a localization query. As distance function $D(V_1^*, V_2^*)$ we can again use the polar coordinate metric, which also satisfies the triangle inequality required above.
- For better results the distance functions can be further modified, or even new distances may be investigated. For example, the translation vector in Section 6.1 or the kernel points in Section 6.2 can also be chosen using other strategies to further improve the quality of the localization. Moreover, the approximate matching

strategies for the Hausdorff distance as well as for other distances may be implemented.
- We want to make the algorithm robust also against small occlusions (e.g., caused by chairs, desks, or small dynamic obstacles). This can be achieved by modifying the distance functions or preprocessing the scans.

Another goal is to implement the matching algorithm (see Section 5), which eventually determines the position of the robot from the scan and the skeleton with highest resemblance to the scan.

Acknowledgements

We gratefully acknowledge helpful comments given by Rolf Klein, Hartmut Noltemeier, Sven Schuierer, and Knut Verbarg. This research is supported by the Deutsche Forschungsgemeinschaft (DFG) under project numbers No. 88/14-1 and No. 88/14-2.

References

[1] H. Alt, B. Behrends, J. Blömer, Approximate matching of polygonal shapes, Ann. Math. Artif. Intell. 13 (1995) 251–265.

[2] E.M. Arkin, L.P. Chew, D.P. Huttenlocher, K. Kedem, J.S.B. Mitchell, An efficiently computable metric for comparing polygonal shapes, IEEE Trans. Pattern Anal. Mach. Intell. 13 (1991) 209–216.

[3] I.J. Cox, Blanche – an experiment in guidance and navigation of an autonomous robot vehicle, IEEE Trans. Robot. Automat. 7 (2) (1991) 193–204.

[4] G. Dudek, K. Romanik, S. Whitesides, Localizing a robot with minimum travel, Proc. 6th Annual ACM–SIAM Symp. on Discrete Algorithms (1995) 437–446.

[5] L.J. Guibas, R. Motwani, P. Raghavan, The robot localization problem, in: K. Goldberg, D. Halperin, J.-C. Latombe, R. Wilson (Eds.), Algorithmic Foundations of Robotics A.K. Peters, 1995, pp. 269–282. http://theory.stanford.edu/people/motwani/postscripts/localiz.ps.Z.

[6] 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.

[7] K. Mehlhorn, S. Näher, LEDA – a platform for combinatorial and geometric computing, Commun. ACM 38 (1995) 96–102. http://www.mpi-sb.mpg.de/guide/staff/uhrig/ledapub/reports/leda.ps.Z.

[8] H. Noltemeier, K. Verbarg, C. Zirkelbach, A data structure for representing and efficient querying large scenes of geometric objects: MB*-trees, in: G. Farin, H. Hagen, H. Noltemeier (Eds.), Geometric Modelling, Computing Supplement, vol. 8, Springer, Berlin, 1993, pp. 211–226.

[9] S. Schuierer, Efficient robot self-localization in simple polygons, in: O. Karch, H. Noltemeier, K. Verbarg (Eds.), 13th European Workshop on Computational Geometry (CG '97), University of Würzburg, 1997, pp. 20–22.

[10] Th. Wahl, Distance functions for polygons and their application to robot localization (in German), Master's Thesis, University of Würzburg, June 1997.

[11] C.M. Wang, Location estimation and uncertainty analysis for mobile robots, in: I.J. Cox, G.T. Wilfong (Eds.), Autonomous Robot Vehicles, Springer, Berlin, 1990.