Robot Localization Using Polygon Distances*

Oliver Karch[1], Hartmut Noltemeier[1], and Thomas Wahl[2]

[1] Department of Computer Science I, University of Würzburg,
Am Hubland, 97074 Würzburg, Germany
{karch,noltemei}@informatik.uni-wuerzburg.de
[2] ATR Interpreting Telecommunications Research Laboratories,
2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan
twahl@itl.atr.co.jp

Abstract. We present an approach to the localization problem, for which polygon distances play an important role. In our setting of this problem the robot is only equipped with a map of its environment, a range sensor, and possibly a compass.
To solve this problem, we first study an idealized version of it, where all data is exact and where the robot has a compass. This leads to the pure geometrical problem of fitting a visibility polygon into the map. This problem was solved very efficiently by Guibas, Motwani, and Raghavan. Unfortunately, their method is not applicable for realistic cases, where all the data is noisy.
To overcome the problems we introduce a *distance function*, the *polar coordinate metric*, that models the resemblance between a range scan and the structures of the original method. We show some important properties of the polar coordinate metric and how we can compute it efficiently. Finally, we show how this metric is used in our approach and in our experimental Robot Localization Program RoLoPro.

1 The Localization Problem

We investigate the first stage of the *robot localization problem* [3,12]: 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. The application we have in mind here is a wake-up situation (e.g., after a power failure or maintenance works), where the robot is placed somewhere in its environment, powered on, and then "wants" to know where it is located. Note, that we do not assume any knowledge about previous configurations of the robot (before its shutdown), because the robot might have been moved meanwhile.

In order to perform 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 a set of range measurements (usually at equidistant angles). The localization should

* This research is supported by the Deutsche Forschungsgemeinschaft (DFG) under project numbers No 88/14-1 and No 88/14-2.

Christensen et al. (Eds.): Sensor Based Intelligent Robots, LNAI 1724, pp. 200–219, 1999.

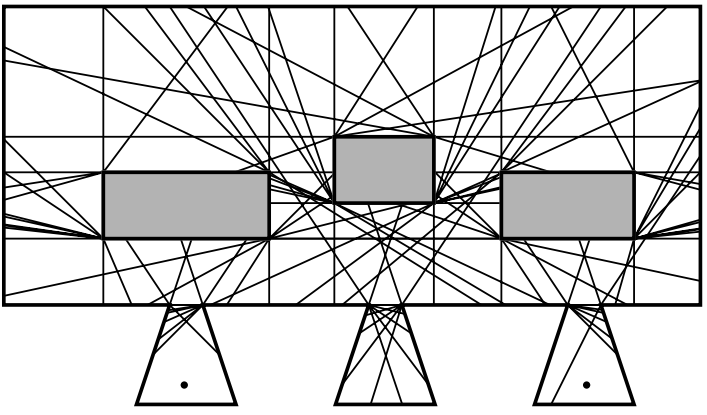

Fig. 1. Polygonal map and its decomposition into visibility cells

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.

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. Figure 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 traveling 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. Its task is 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 [10].

This paper concentrates on the first stage of the localization process, that is, on generating the possible robot configurations (i.e., positions and orientations), although in our current work we also want to give solutions for the second stage, which can be applied in practice. 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 $\mathcal{M}$ and a star-shaped polygon $\mathcal{V}$ (the visibility polygon of the robot), find all points $p \in \mathcal{M}$ that have $\mathcal{V}$ as their visibility polygon.

Guibas et al. [5] described a scheme for solving this idealized version of the localization problem efficiently and we will briefly sketch their method in the following section. 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 Sect. 3 we consider these problems and show in Sections 4 and 5 an approach to avoiding them, which uses *distance functions* to model the resemblance between the noisy range scans (from the sensor) and the structures of the original method (extracted from the possibly inexact map).

2 Solving the Geometric Problem

In the following we sketch the method of Guibas et al., which is the basis for our approach described in Sections 4 and 5. 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 $\mathcal{M}$, called the *map polygon*. Additionally, we assume that $\mathcal{M}$ 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 $p \in \mathcal{M}$ such that their visibility polygon $\mathcal{V}_p$ is identical with the visibility polygon $\mathcal{V}$ of the robot.

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[1] 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 $\mathcal{M}$ is divided into convex *visibility cells* by introducing straight lines forming the boundary of the cells such that the following property holds:

[1] "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.

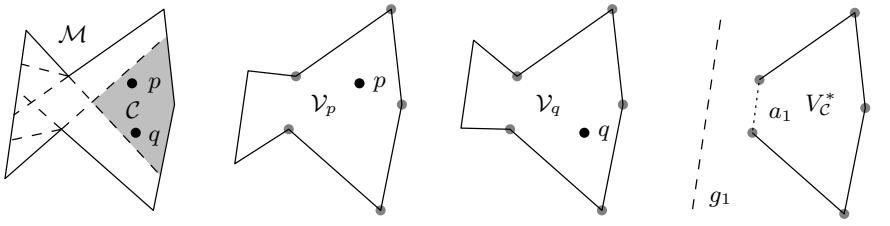

Fig. 2. Decomposition of a map polygon into visibility cells (left), two visibility polygons (middle), and the corresponding skeleton (right)

> 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 $\geq \pi$), the 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 $\mathcal{C}$ see the same set of five map vertices (marked gray in the corresponding visibility polygons in the middle). Figure 1 shows a decomposition for a more complex map with three obstacles (gray), generated with our software RoLoPro described in Sect. 6.

If the map consists of a total number of n vertices, of which r are reflex, the number of introduced rays is in $\Theta(nr)$ in the worst-case. Therefore the worst-case complexity of the decomposition is in $\Theta(n^2r^2)$. For map polygons without holes it can be shown that this complexity is actually in $\Theta(n^2r)$. 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 observe 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 spurious edges and the spurious 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* $V_{\mathcal{C}}^*$ as the common skeleton of all visibility polygons of points from cell $\mathcal{C}$. Figure 2 shows an example of the common skeleton $V_{\mathcal{C}}^*$ of two visibility polygons $\mathcal{V}_p$ and $\mathcal{V}_q$ for viewpoints p and q from the same cell $\mathcal{C}$.

2.3 Performing a Localization Query

When we construct the skeleton V_p^* from the visibility polygon $\mathcal{V}_p$, 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. It is already shown by Guibas et al. that exactly those points q are valid robot positions for a given visibility polygon $\mathcal{V}_p$ that have the following two properties:

1. The point q lies in a visibility cell $\mathcal{C}$ with $V_{\mathcal{C}}^* = V_p^*$.
2. The position of q relative to $V_{\mathcal{C}}^*$ is the same as the position of p relative to V_p^*.

The consequence is that in order to determine all points in the map that have $\mathcal{V}_p$ as their visibility polygon, it suffices to consider the equivalence class of all cells with visibility skeleton V_p^* (first property) and then to determine the subset of cells consisting of those that contain a viewpoint with the same relative position to V_p^* as p (second property). After that, all remaining viewpoints are valid robot positions.

Hence, for performing the localization query, the skeleton V_p^* of the given visibility polygon $\mathcal{V}_p$ is computed and the corresponding equivalence class of skeletons 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 $\mathcal{C}$ relative to its cell skeleton $V_{\mathcal{C}}^*$, we can easily determine for each cell in the equivalence class the corresponding viewpoint of the robot and check whether it lies in $\mathcal{C}$. This test can be performed very efficiently by using a point location structure; in fact, using a sophisticated preprocessing of the visibility cells only a *single point location query* is necessary in order to test the viewpoints of *all cells* in an equivalence class at once.

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

The total preprocessing time and space of this approach is in $\mathcal{O}(n^2 r \cdot (n + r^2))$ for map polygons without holes (see [7]).

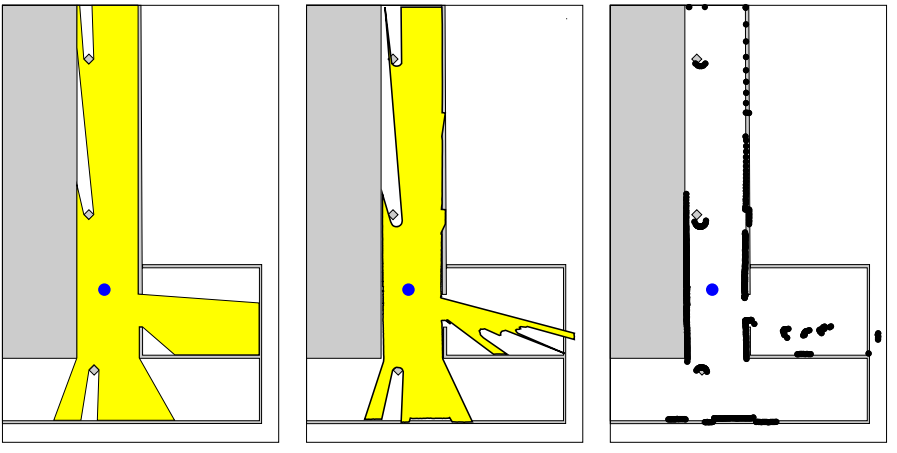

Fig. 3. Exact visibility polygon (left) and approximated visibility polygon (middle) of a noisy scan (right)

3 Problems in Realistic Scenarios

The idealizing assumptions of the method described in Sect. 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. 3, which shows in the left part an exact visibility polygon of a robot (based on a map of our department). In the right part of the figure we see a real noisy laser range scan taken at the corresponding position in our department using a SICK LMS 200 laser scanner. Even if we connect the scan points by straight line segments as shown in the middle part of the figure, we only get an approximation $\mathcal{V}_\mathcal{S}$ of the exact visibility polygon $\mathcal{V}$ based on the map.
- 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.
- 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. Such obstacles can also be recognized in the right part of the example scan of Fig. 3.
- 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 $V^*_\mathcal{S}$, which the robot computes from its approximated visibility polygon $\mathcal{V}_\mathcal{S}$, 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.

4 Adaptation to Practice

Our approach to tackling these problems is, for a given range scan $\mathcal{S}$ (from the sensor), to search for the preprocessed skeleton that is *most similar* to the scan. For modeling the resemblance between a scan $\mathcal{S}$ and a skeleton V^* we use an appropriate *distance function* $d(\mathcal{S}, V^*)$. 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 $d(\mathcal{S}, V^*)$ to find the skeleton with the highest resemblance to the scan.

Depending on the distance function, we then additionally have to apply a local 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.

4.1 Requirements to the Distance Function

In order to be useful in practice, a distance function $d(\mathcal{S}, V^*)$ 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 $d_{\mathcal{S}}(\mathcal{S}_1, \mathcal{S}_2)$ and $d_{V^*}(V_1^*, V_2^*)$ be functions that measure the resemblance between two scans $\mathcal{S}_1$ and $\mathcal{S}_2$ and between two skeletons V_1^* and V_2^*, respectively. An appropriate reference distance measure for $d_{\mathcal{S}}(\mathcal{S}_1, \mathcal{S}_2)$ and $d_{V^*}(V_1^*, V_2^*)$ is, for example, the Hausdorff distance (see Sect. 5.1).

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

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

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

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

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

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 $\mathcal{S}$ should have a small distance value $d(\mathcal{S}, V^*)$. Otherwise, the distance would not give any advice for finding a well-matching skeleton and therefore would be useless for the localization algorithm. In particular, if we take a scan $\mathcal{S}$ from a point p whose skeleton equals V^*, we want the distance $d(\mathcal{S}, V^*)$ 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 Sect. 4.2), the computation costs should not be too high.

4.2 Maintaining the Skeletons

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 Sect. 2.1), the skeletons should be stored in an appropriate data structure that we can search through efficiently.

For this purpose we can use the *Monotonous Bisector Tree* [9], 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 a 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*

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

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.

5 Suitable Distances for $d(\mathcal{S}, V^*)$ and $D(V_1^*, V_2^*)$

It is hard to find distance functions that have all the properties from Sect. 4.1. Particularly, the fifth requirement (fast computability) 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. And of course, it is even more difficult to find for a given scan-skeleton distance $d(\mathcal{S}, V^*)$ a compatible distance $D(V_1^*, V_2^*)$, which we need for performing the nearest-neighbor query efficiently.

In the following, we investigate two distance functions, the Hausdorff distance and the polar coordinate metric, and illustrate the occurring problems.

5.1 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 $\mathcal{T}$ 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

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

It can easily be shown that the minimum Hausdorff distances are continuous and by definition also fulfill the third and forth property of Sect. 4.1. 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 $\mathcal{S}$ 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 $\mathcal{C}_{V^*}$ (e.g., the center of gravity of $\mathcal{C}_{V^*}$). This is reasonable, because by the definition of the visibility cells, exactly the points in $\mathcal{C}_{V^*}$ 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 $d(\mathcal{S}, V^*)$ now does not only depend on V^*, but also on the visibility cell itself.[2] 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 $t \in \mathcal{T}$ with $\delta(A, t(B)) \leq c \cdot \delta_{\min}(A, B)$, for a constant $c \geq 1$. 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 $\vec{\delta}(V^*, \mathcal{S})$ and $\vec{\delta}_{\min}(V^*, \mathcal{S})$ almost always return an infinite 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 $\vec{\delta}(\mathcal{S}, V^*)$ and $\vec{\delta}_{\min}(\mathcal{S}, V^*)$. Note that if we pursue the second approach, the distance $\vec{\delta}_{\min}(\mathcal{S}, V^*)$ 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.

5.2 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 [11], which takes

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

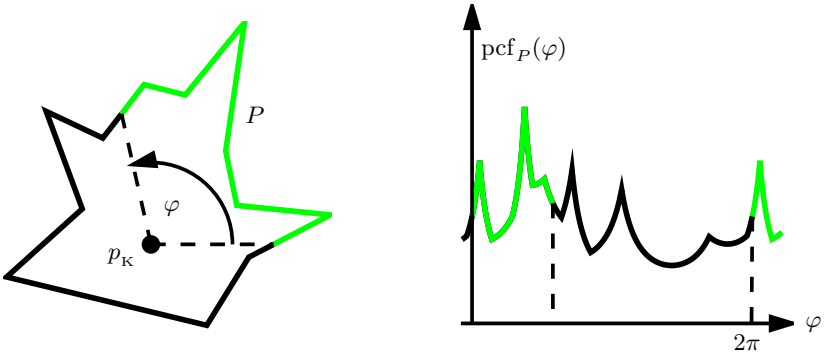

Fig. 4. The polar coordinate function $\mathrm{pcf}_P(\varphi)$ for a star-shaped polygon P

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 $\mathcal{V}_S$ (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 $\mathcal{C}_{V^*}$ all full edges are completely visible, and for each artificial edge a_i a part of the corresponding straight line g_i is visible.

To define 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 for short)

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

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. Figure 4 depicts the PCF 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):

$$\mathrm{pcm}(P,Q) := \min_{t \in [0,2\pi[} \sqrt{\int_0^{2\pi} \left(\mathrm{pcf}_P(\varphi - t) - \mathrm{pcf}_Q(\varphi)\right)^2 \mathrm{d}\varphi} \qquad (1)$$

Figure 5 shows an example, where the two graphs are already translated in a way such that this integral norm is minimized.

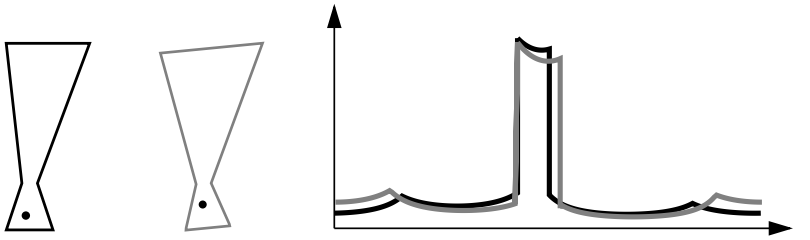

Fig. 5. Computation of the PCM as minimum integral norm

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 Sect. 4.1 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 fulfills the continuity requirement of Sect. 4.1, 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, for example, the left-most kernel point does *not* depend continuously on the polygon.

Wahl [11] also 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 $t \in \mathcal{T}$, the equality $t(p_{\mathrm{K}}) = p'_{\mathrm{K}}$ must hold, for all polygons P and all $t \in \mathcal{T}$. For example, the center of gravity of the kernel of the polygon has this property.

Using the PCM as distance $d(\mathcal{S}, V^*)$ If we want to use the PCM as a distance function $d(\mathcal{S}, V^*)$ we need corresponding star-shaped polygons for $\mathcal{S}$ and V^* that can be used as polygonal representatives for the scan and the skeletons:

- For the scan, we choose the approximated visibility polygon $\mathcal{V}_\mathcal{S}$, 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 $\mathcal{C}_{V^*}$), we choose a point c inside the cell $\mathcal{C}_{V^*}$ (e.g., the center of gravity) and determine the visibility polygon $\mathcal{V}_c$ of this point. By construction, c is a kernel point of $V^*_\mathcal{C}$.

In the sequel we will only use $\mathcal{V}_\mathcal{S}$ and $\mathcal{V}_c$ for determining the distance measures. Then, our goal is to find the polygon $\mathcal{V}_c$ that is most similar to the approximated visibility polygon $\mathcal{V}_\mathcal{S}$ with respect to $\mathrm{pcm}(\mathcal{V}_\mathcal{S}, \mathcal{V}_c)$. With this choice we obtain the following theorem about the polar coordinate metric as a distance function:

Theorem 1. *The distance function* $d(\mathcal{S}, V^*) := \mathrm{pcm}(\mathcal{V}_\mathcal{S}, \mathcal{V}_c)$*, with* $\mathcal{V}_\mathcal{S}$ *and* $\mathcal{V}_c$ *as defined above, fulfills the following requirements from Sect. 4.1: continuity and invariance against translations and rotations.*

Note that the PCM is not similarity preserving: if the point c chosen above for computing a corresponding polygon for a visibility cell $\mathcal{C}$ 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 $p \in \mathcal{C}$, the distance from p to the corresponding point $c \in \mathcal{C}$ is not too large. Thus, the approximated visibility polygon $\mathcal{V}_\mathcal{S}$ and the visibility polygon $\mathcal{V}_c$ differ not too much, and the value of $\mathrm{pcm}(\mathcal{V}_\mathcal{S}, \mathcal{V}_c)$ is small.

Computing the PCM value efficiently The exact computation of the minimum in (1) seems to be difficult and time consuming, since the polar coordinate functions of a polygon with p edges consists of p pieces of functions of the form $c_i / \sin(\varphi + \alpha_i)$. For one fixed translation t (this corresponds to the case, when the robot already knows its exact orientation), the integral

$$\sqrt{\int_0^{2\pi} \left(\mathrm{pcf}_P(\varphi - t) - \mathrm{pcf}_Q(\varphi)\right)^2 \,\mathrm{d}\varphi}$$

can be computed straightforward in linear time $\mathcal{O}(p+q)$, where p and q stand for the complexities of the two polygons. But the global minimum over all possible values of $t \in [0, 2\pi[$ seems to be much harder to determine.

Therefore, we use two different approximative approaches for computing a suitable PCM value. Both approaches use a set of supporting angles/points for each of the two involved polar coordinate functions. For a given polygon its supporting angles are the angles that correspond to a vertex of the polygon plus the angles of the local minima of the inverse sine functions (see the left part of Fig. 6). The ideas of the two approximative approaches are then as follows:

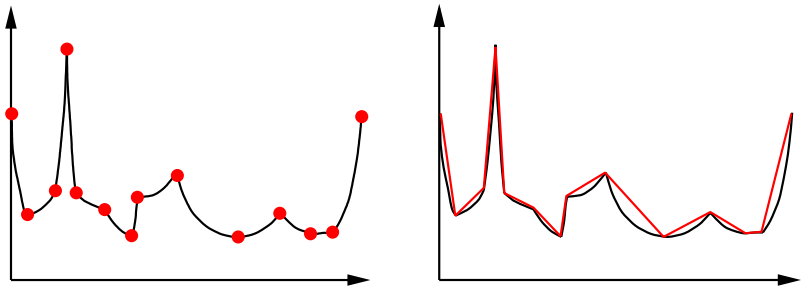

Fig. 6. Two approaches for an approximative PCM value

1. For the first approach we concentrate on the $\mathcal{O}(p)$ (or $\mathcal{O}(q)$, respectively) supporting angles (see the left part of Fig. 6). Namely, we do not minimize over *all rotation angles* of the two polygons, but only over the $\mathcal{O}(pq)$ rotation angles, that place one supporting angle of the first polygon on a supporting angle of the second one. The integral values are then computed exactly in time $\mathcal{O}(p+q)$. Summing up, we need $\mathcal{O}(pq(p+q))$ time to compute this approximated value of the PCM.
2. In the second approach we compute an *exact minimum* over all rotation angles, but use a modified polar coordinate function. Namely, we introduce a linear approximation of the PCM, which also has all metric properties and is sufficient for our applications. This approximation is depicted in the right part of Fig. 6: the supporting points (that correspond to a polygon vertex or a local minima of the PCF) are connected by straight line segments to get a modified PCF. The minimum integral norm is then defined like in the non-approximated version of the PCM (see (1)). 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 $\mathcal{O}(pq \cdot (p+q))$. Of course, if we do not want to minimize over the rotations, the computation time is again in $\mathcal{O}(p+q)$ like for the non-approximated version.

Using the PCM as distance $D(V_1^*, V_2^*)$ Since the PCM has all metric properties, it particularly fulfills the triangle inequality. Therefore, we can use it not only for defining the scan-skeleton distance $d(\mathcal{S}, V^*)$, but also for defining a *compatible* skeleton-skeleton distance $D(V_1^*, V_2^*)$.

For this task, we again use for each pair of a skeleton V^* and its corresponding cell $\mathcal{C}_{V^*}$ the polygonal representative $\mathcal{V}_c$ as defined in the last section. Then, the triangle inequality

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

follows immediately from the triangle inequality of the PCM,

$$\mathrm{pcm}(\mathcal{V}_{\mathcal{S}}, \mathcal{V}_{c_2}) \leq \mathrm{pcm}(\mathcal{V}_{\mathcal{S}}, \mathcal{V}_{c_1}) + \mathrm{pcm}(\mathcal{V}_{c_1}, \mathcal{V}_{c_2}) \ ,$$

and we can apply the Monotonous Bisector Tree to the set of skeletons for increasing the performance of the nearest-neighbor query.

The PCM in realistic scenarios Since the PCM is continuous and invariant under Euclidean transformations, we can hope that the first two problems mentioned in Sect. 3 (the noisy scans and the unknown robot orientation) are solved satisfactorily.

Also the third problem (unknown obstacles in the environment) does not strongly influnce the PCM as long as the obstacles take up only a small interval of the whole scanning angle. But the other case, where the obstacles occupy a large part of the robot's view, poses a problem to our localization method. Here, additional (heuristic) algorithms are needed to detect such cases.

In contrast to this, a possibly limited sensing range (the fourth problem addressed in Sect. 3) can be easily tackled in a straightforward manner: when we compute the distances $d(\mathcal{S}, V^*)$ and $D(V_1^*, V_2^*)$, we simply cut off all distance values larger than the sensing range, that is, for all occuring polygons we use a modified PCF,

$$\mathrm{pcf}'_P(\varphi) := \min\{\mathrm{pcf}_P(\varphi), \text{sensing range}\} \ .$$

6 Our Implementation RoLoPro

We have implemented the two versions of the localization algorithm in C++ using the LEDA Library of Efficient Datatypes and Algorithms [8], namely the original method described in Sect. 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. Figure 7 shows some screen shots of our robot localization program ROLOPRO processing localization queries in real as well as in simulated environments.

As distance function $d(\mathcal{S}, V^*)$ we have implemented the Hausdorff distance and the polar coordinate metric described in Sect. 5. Only for the PCM the efficient skeleton management described in Sect. 4.2 was implemented, since for the Hausdorff distance we could not find a suitable distance $D(V_1^*, V_2^*)$. Therefore, in this case the scan has to be compared with *all skeletons*, which is

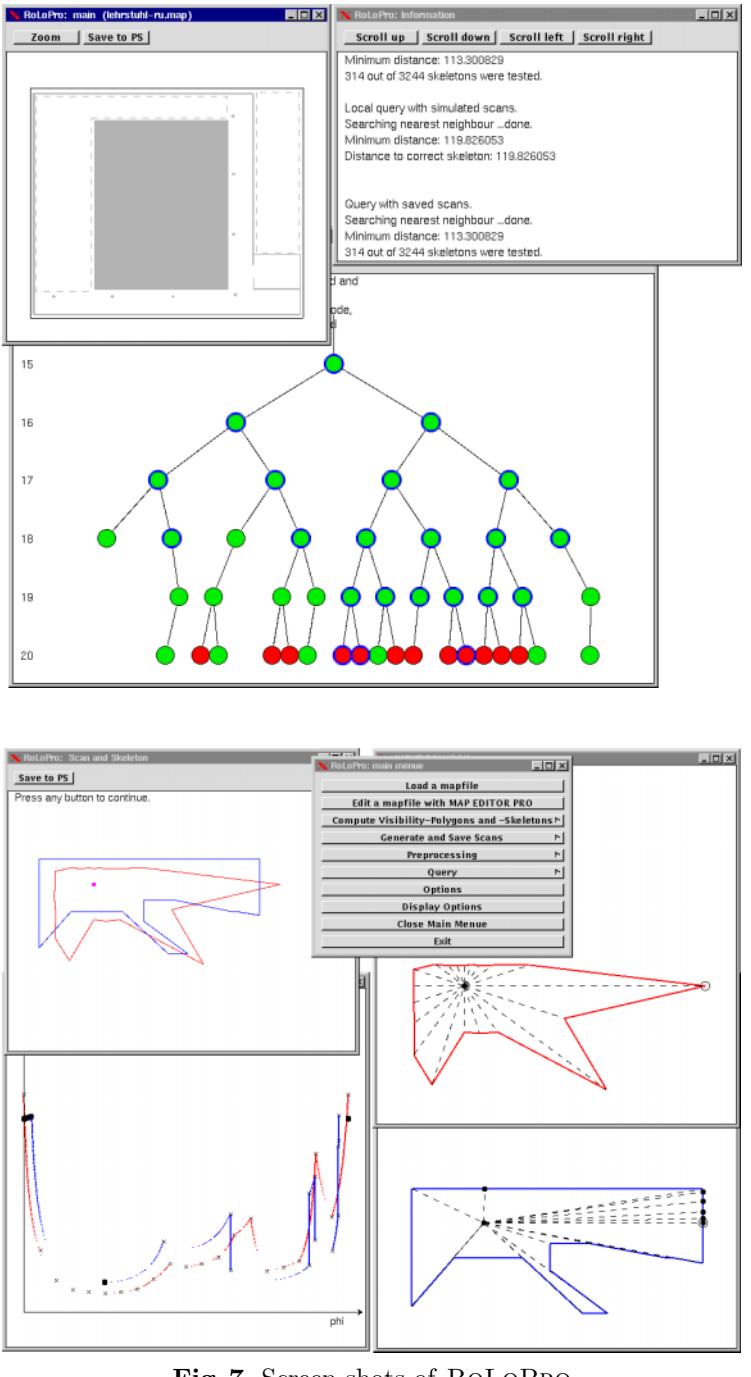

Fig. 7. Screen shots of RoLoPro

much more time consuming than the PCM approach, which uses the Monotonous Bisector Tree.

Furthermore, we have implemented some additional features into RoLoPro, described in the following.

Noisy compass We are able to model different kinds of compasses by using modified *minimization intervals* in (1), namely

- an exact compass: here the minimization interval consists only of a single value that represents the orientation of the robot with respect to the map,
- no compass at all: since we know nothing about the robot's orientation, we use $[0, 2\pi[$ as the minimization interval,
- a noisy compass: here we use a smaller interval $[o - \varepsilon, o + \varepsilon[\subset [0, 2\pi[$ that depends on the orientation of the robot and the uncertainty $\varepsilon < \pi$ of the compass.

In most cases we have to carefully adapt the distances $d(\mathcal{S}, V^*)$ and $D(V_1^*, V_2^*)$ in different ways, such that the requirements from Sect. 4.1 and the triangle inequality remain fulfilled.

Partial range scans By modifying the *integration interval* in (1) we also can process partial range scans, where the scanning angle is less than 2π. As well as in the noisy-compass feature the distances have to be carefully modified, in particular if we want to use both features at the same time.

Additional coarsening step The complexity of the visibility cell decomposition, which is necessary for the geometrical approach described in Sect. 2 and which is the basis for our own approach, is quite high. But in practice many of the visibility cells need not be considered, because their skeletons differ only slightly from the skeletons of neighboring cells and the cells themselves are often very small.

Therefore, we have implemented an additional coarsening step, where neighboring cells with small distances $D(V_1^*, V_2^*)$ are combined into one bigger cell. This way, we need less space for storing the cells as well as less time for constructing the Monotonous Bisector Tree. Of course, this coarsening procedure reduces the quality of the localization, since we obtain both, a fewer number of distance values and larger visibility cells. Thus, the threshold value for the merging step must be carefully chosen by the user.

7 First Experimental Tests

First tests in small simulated scenes have shown a success rate of approximately 60 % for the directed Hausdorff distance, i.e., the scan origins of about 60

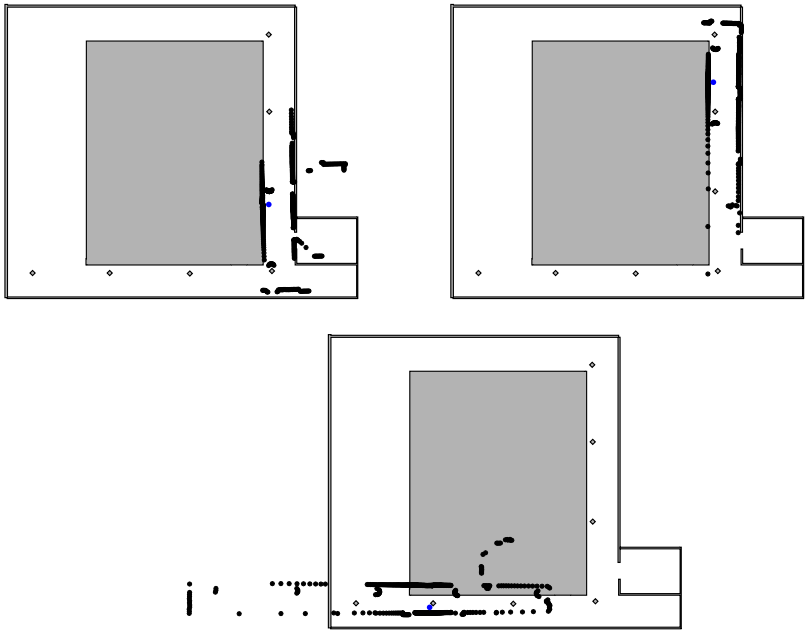

Fig. 8. Some localization examples using the PCM

out of 100 randomly generated scans were inside that cell with the smallest distance $d(\mathcal{S}, V^*)$ to the scan. In the same scenes the success rate of the polar coordinate metric was about 80 - 90 %.

We also have evaluated our approach in real environments (using the PCM as distance function), namely in our department and in a "less friendly" supermarket environment, where the scans were very noisy. In both cases the scans were generated by a SICK PLS/LMS 200 laser scanner and we always assumed that the robot has a compass.

Figure 8 shows some localization examples for the first environment: we used a partial map of our department, which consists of 76 vertices, 54 of them reflex. This led to a total number of about 4 300 visibility cells, which were finally reduced to about 3 300 cells by our coarsening step (using a suitable threshold). Each of the full range scans consists of 720 range measurements.

Almost all localization queries resulted in solutions like the first two examples in Fig. 8. Only about 5 - 10 % of the localization queries failed. For example, the reason for the bad result of the third query in Fig. 8 probably were some scan points at a distance of about 50 m in the right part of the scan (cut off in the figure), which were produced by scanning through a window. Since the PCM "tries" to minimize a kind of *average* quadratic error, the scan was shifted in the opposite direction to the left.

In the supermarket environment our current localization approach completely failed, probably because of the very noisy range scans, which produced effects

like the third example of Fig. 8. Moreover, in this environment only half range scans were available such that the localization problem gets even harder.

8 Future Work

Our main goals for the future are to overcome some of the current limitations of our approach:

- We try to further reduce the large space requirements of our approach (caused by the space consuming visibility cell decomposition). For this task we want to use a more sophisticated approach than our current coarsening step, for example by considering also the scan resolution and the limited sensing range.
- Currently, the performance of our approach is not good enough, if we have no compass information, because the distance computation for this case takes $\mathcal{O}(pq(p+q))$ time, instead of linear time for the case of an exact compass. Therefore, we want to speed up the distance computation using a suitable scan preprocessing.
- For the case of extremely disordered scans we get very bad localization results (like in the supermarket environment described above). To overcome this problem, we try to modify the PCM appropriately and try to do an additional preprocessing for the scans.

Another goal is to implement a matching algorithm (see Sect. 4), which eventually determines the position of the robot from the scan and that skeleton with highest resemblance to the scan. Otherwise, we would only know the *visibility cell*, where the robot is located, that is, a possibly coarse approximation of its position.

Furthermore, we also want to integrate navigation algorithms into our approach, such that the robot can move around to eliminate ambiguous positions.

Acknowledgements

We gratefully acknowledge helpful comments given by Rolf Klein, 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, and J. Blömer. Approximate Matching of Polygonal Shapes. *Annals of Mathematics and Artificial Intelligence*, 13:251–265, 1995.
2. E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell. An Efficiently Computable Metric for Comparing Polygonal Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:209–216, 1991.

3. I. J. Cox. Blanche — An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle. *IEEE Transactions on Robotics and Automation*, 7(2):193–204, April 1991.
4. G. Dudek, K. Romanik, and S. Whitesides. Localizing a Robot with Minimum Travel. *SIAM Journal on Computing*, 27(2):583–604, 1998.
5. 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. 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. O. Karch and Th. Wahl. Relocalization — Theory and Practice. *Discrete Applied Mathematics (Special Issue on Computational Geometry), to appear*, 1999.
8. 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.
9. H. Noltemeier, K. Verbarg, and C. Zirkelbach. A Data Structure for Representing and Efficient Querying Large Scenes of Geometric Objects: MB*–Trees. In G. Farin, H. Hagen, and H. Noltemeier, editors, *Geometric Modelling*, volume 8 of *Computing Supplement*, pages 211–226. Springer, 1993.
10. S. Schuierer. Efficient Robot Self-Localization in Simple Polygons. In R. C. Bolles, H. Bunke, and H. Noltemeier, editors, *Intelligent Robots – Sensing, Modelling, and Planning*, pages 129–146. World Scientific, 1997.
11. Th. Wahl. Distance Functions for Polygons and their Application to Robot Localization (in German). Master's thesis, University of Würzburg, June 1997.
12. C. M. Wang. Location estimation and uncertainty analysis for mobile robots. In I. J. Cox and G. T. Wilfong, editors, *Autonomous Robot Vehicles*. Springer, Berlin, 1990.