Learning the Delaunay Triangulation of Landmarks From a Distance Ordering Sensor

Max Katsev and Steven M. LaValle

***Abstract*— This paper considers a robot that moves in a plane and is only able to sense the distance order of landmarks with respect to its current position. The robot has no access to either metric information about the location of landmarks and its own position, or to odometry or speed controls. We propose several algorithms for the robot that allow it to navigate to certain points in the plane and to learn global information about the landmark locations. We, furthermore, demonstrate example tasks, such as convex hull computation, that can be performed using this information.**

I. Introduction

In this paper we study the navigation problem for a robot with very limited sensing: for any two landmarks out of the pre-defined set, the robot can only detect which one it is closer to. The question that we ask is what information can the robot learn and what tasks can it perform using this information. Is it enough to perform navigation? How can we even define meaningful tasks in the absence of reliable position information?

Landmarks are usually defined as distinctive stationary features or objects that the robot can detect using its sensors. Significant amount of research has been done that uses landmarks for navigation [1], [2], [3]. Landmarks have been used to infer the robot position, for example, by triangulation [4], [5]. Alternatively, a geometric map of the environment can be built by using the simultaneous localization and mapping (SLAM) approach [6], [7].

This paper follows a minimalist approach to sensing, which means that we are trying to keep the model of robot sensors and motion primitives as simple as possible [8], [9], [10]. By doing so, we avoid the difficult task of constructing a complete model of environment and performing the full estimate of the robot state. Instead, we concentrate on creating a *topological map* [1], [11], [12], [13] that encodes information the robot can learn using its weak sensors. Our research aims to answer the basic question of what data can be gathered by a sensor and what is the relationship between the sensors, motion primitives and the representation of the environment that the robot can create [14], [15], [16].

Our work has been heavily influenced by [17], which explores similar idea in a different setting. In this paper, the robot's sensor produces the *distance order* of the landmarks. In [17], the most advanced of the robot's few sensors, instead, gives the *cyclic order* of landmarks around the robot. Such a sensor allows the robot to detect when it crosses the line that goes through any pair of landmarks. The authors describe

M. Katsev and S. M. LaValle are with the Department of Computer Science, University of Illinois, Urbana, IL 61801, USA {katsev1,lavalle}@uiuc.edu

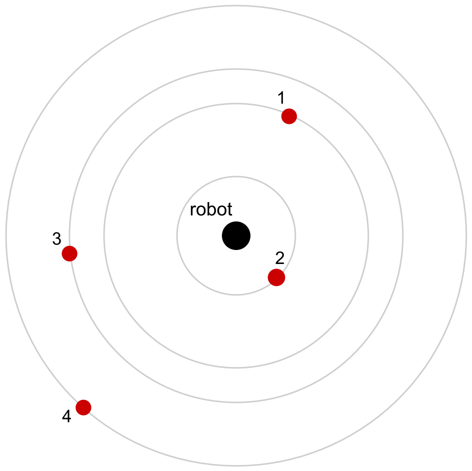


Fig. 1. The sensor gives the list of landmarks in order of increasing distance from the robot, but not the actual distances. In this example, the sensor output would be [2,1,3,4]

a variety of tasks that the robot can perform: learning the relationship between the locations of multiple landmarks, patrolling a set of landmarks, and, finally, navigation.

We show that despite the weak sensing capabilities, the robot is able to reliably reach certain key locations in the environment and learn the Delaunay triangulation of the set of landmarks. The Delaunay triangulation encodes information about landmarks' relative positions [18, ch. 9] and has been used in various path planning approaches [19], [20], [21]. In particular, the minimum spanning tree is a subtree of the Delaunay triangulation [22] and the shortest path distance in the Delaunay triangulation graph is a constant factor approximation of the Euclidean distance [23].

The paper is structured as follows. Section II describes the basic model for robot sensing and actuation. Section III presents algorithms for learning the Delaunay triangulation. Section IV provides examples of how the triangulation can be used to perform certain tasks. Finally, Section V describes open questions related to the present research.

II. Model

A. Basic Model

The robot is considered to be a moving point in $\mathbb{R}^2$. For the purposes of this paper we will assume that there are no obstacles and the entire plane is free space. Let $L = \{l_1, \dots, l_n\} \subset \mathbb{R}^2$ be a set of n special points (or *landmarks*); their locations are unknown to the robot. The landmarks are distinguishable and have unique labels. We assume that the landmarks are in general position: no three landmarks are

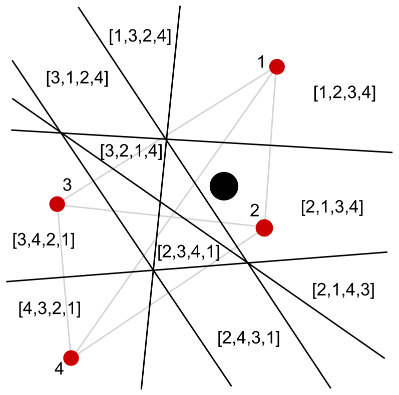


Fig. 2. Perpendicular bisectors split the environment into polygonal preimages. Each preimage can be uniquely identified by the sensor output when the robot is inside the region. Some preimage labels are omitted for clarity

collinear, no four landmarks are cocircular, and no landmark is equidistant from two other landmarks.

Let $r \in \mathbb{R}^2$ be the current robot position. The *state space* for the robot is defined as

$$X = \mathbb{R}^2 \times \mathcal{E},$$

in which $\mathcal{E} \subset \mathbb{R}^{2n}$ is a set of all possible landmark configurations.

The robot is equipped with a *distance ordering sensor* that returns the sequence of landmark labels ordered by the distance relative to the current robot position in ascending order (see Fig. 1) and additional information about landmarks that are equidistant from the robot. Such a sensor can be implemented, for example, by comparing the strength of radio signals transmitted by the landmarks.

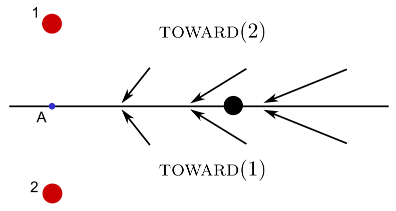
The robot is able to execute two motion primitives: $\text{TOWARD}(l)$ and $\text{AWAY}(l)$. $\text{TOWARD}(l)$ results in the robot moving in a straight line in the direction of the landmark labeled l, whereas $\text{AWAY}(l)$ moves the robot in the opposite direction ($\text{AWAY}(l)$ is undefined when the robot is at l). Additionally, $\text{TOWARD}(l)$ terminates when the robot arrives at l. Note that these motion primitives do not provide the robot with any information about the relationship between directions to multiple landmarks.

B. Additional Considerations

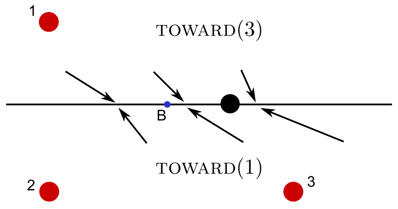
The distance ordering sensor can be defined by a sensor mapping

$$h : X \to S_n \times \mathcal{P}_{n-1},$$

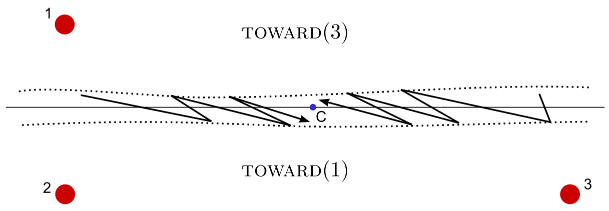
in which S_n is a set of all possible permutations of n distinct elements and $\mathcal{P}_{n-1}$ is the powerset of $\{1, \dots, n-1\}$. The first part of the output lists the landmarks in the order of ascending distance to the robot. The second part encodes groups of landmarks with equal distances to the current robot location: if it contains index i, then distances to the i-th and $i+1$-th closest landmarks are equal. For example, if $h(r, E) = ((l_4, l_2, l_1, l_3), \{1, 2\})$, then, in the configuration E, the distances from r to l_4, l_2, and l_1 are equal and smaller than the distance to l_3.



(a) Projections of both direction fields on the bisector have the same direction. The robot will move in that direction until it reaches point A



(b) Projections of both direction fields on the bisector have opposite directions. The robot behavior is not clearly defined.



(c) We resolve the ambiguity by introducing a sensor reaction delay (dotted curves show where the sensor output changes after the robot has crossed the bisector). The robot will move in the direction corresponding to the larger projection of the direction field on the bisector (in the direction of point C, in this example).

Fig. 3. Various robot behavior at the boundary between preimages

Alternatively, the sensor output can be encoded as a linear order relation “closer than or equidistant to” on the set of landmarks L that can be represented as an $n \times n$ matrix with elements ± 1 that satisfies certain properties (for example, one of them is to be antisymmetric).

This mapping decomposes the plane into *preimages*: regions in which the sensor output is constant. For the distance ordering sensor, the preimages are polygons created by the perpendicular bisectors (simply *bisectors* in the future) of pairs of the landmarks (see Fig. 2), the bisectors themselves, and the intersection points of multiple bisectors.

The robot can only change its behavior when its sensor readings change, specifically, when it crosses one of the bisectors (we assume that the robot neither controls nor knows its speed; therefore, the notion of *time* is meaningless in our setting). We use the term *direction field* to describe a vector field that assigns to each point of the plane the direction of the robot movement at that point. Due to the specifics of the motion primitives available to the robot, the direction field varies smoothly inside every preimage (with the possible exception of landmark locations). However, it might be problematic to define the robot direction on the boundary between multiple preimages (see Fig. 3).

In the situation depicted in Fig. 3(a), it is reasonable to conclude that the robot will continue moving along the bisector line in the general direction of the landmarks until it reaches point A, then stop; Fig. 3(b) is more ambiguous. To resolve this issue, we will refer to a somewhat different robot model in which the distance ordering sensor has bounded *polling rate*. Effectively, this means that there is a (bounded) delay between the robot crossing the bisector line and the sensor reading being updated. As a result, the actual robot trajectory will look similar to Fig. 3(c): in general, the robot direction is determined by the direction field that has a larger projection on the bisector. In the example of Fig. 3(c), the robot moves along the bisector towards point C. This allows us to make the following statement:

Proposition 1 (Extending the direction field to the preimage boundaries)**.** *Let q be a point on the bisector line that separates preimages P_1 and P_2. Let d_1 and d_2 be continuous extensions to q of the robot direction field defined on P_1 and P_2, correspondingly.*

1) *If both d_1 and d_2 are outward-pointing for P_1 and P_2, then the robot direction at q is defined by the projection of $d_1 + d_2$ on the bisector line.*
2) *If d_1 is outward-pointing and d_2 is inward-pointing, then the robot direction at q is defined by d_1.*
3) *If both d_1 and d_2 are inward-pointing, then the robot direction at q is undefined.*

It is worth remembering that the speed of the robot is not taken into account, so the norm of the direction vector is irrelevant.

III. Learning the Delaunay Triangulation

The following notation will be used in this section:

- $d : \mathbb{R}^2 \times \mathbb{R}^2 \to [0, \infty)$ is the distance function on the plane;
- $\textsc{last}(r)$ is the most distant landmark (last in the sensor output), when the robot is at r, defined by

$$\textsc{last}(r) = \arg\max_{l \in L} d(r, l).$$

A. Distinguishing Acute, Obtuse and Right Angles

The first task the robot is able to perform is to gather some information about the angle formed by three landmarks. The algorithm[1] is very simple:

Algorithm 1. *(Identifying the type of an angle $\angle ABC$)*

Description. The robot executes $\textsc{toward}(A)$, then $\textsc{toward}(C)$. The robot stops when A and C switch places in the distance ordering. If B is the most distant of the three landmarks, then $\angle ABC < \frac{\pi}{2}$; if B is the closest, then $\angle ABC > \frac{\pi}{2}$; finally, if all three landmarks are equidistant, then $\angle ABC = \frac{\pi}{2}$. $\square$

Lemma 1. *Algorithm 1 is correct.*

Proof. It can be seen that the robot stops exactly at the midpoint of the segment AC. The result follows from simple geometric reasoning. $\square$

[1]It should be noted that most of the "algorithms" of this section are not algorithms in the classical sense of the word, but rather motion strategies that instruct the robot how to get to a desired location, without the knowledge of the starting state.

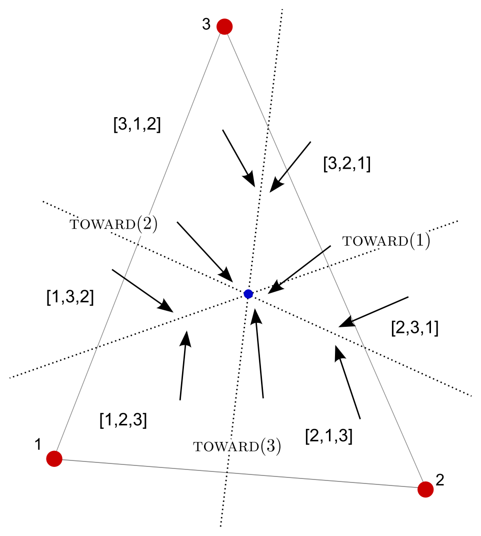


Fig. 4. Locating the circumcenter of an acute triangle.

B. Locating the Circumcenter of Three Landmarks

The crucial part of the Delaunay triangulation algorithm is to navigate the robot to the *circumcenter* of three given landmarks. The way to accomplish this task depends on the type of the triangle formed by those landmarks. Assume for simplicity that A, B and C are the only three landmarks.

Algorithm 2. *(Reaching the circumcenter of a right triangle ΔABC)*

Description. The robot executes $\textsc{toward}(A)$, then $\textsc{toward}(C)$. It stops when A and C switch places in the distance ordering. $\square$

Lemma 2. *If $\angle ABC = \frac{\pi}{2}$, then Algorithm 2 terminates with the robot at the circumcenter of ΔABC.*

Proof. The circumcenter of a right triangle is the midpoint of its hypotenuse, in this case AC. $\square$

Algorithm 3. *(Reaching the circumcenter of an acute triangle ΔABC)*

Description. The robot is continuously executing $\textsc{toward}(\textsc{last}(r))$, in which r is its current position. It stops when the distances to all three landmarks A, B, and C are equal. Notice that due to the iterative nature of the algorithm, the robot does not reach the stopping point perfectly, but only in the limit. We might decide that the robot stops when it arrives to the desired location with good enough precision or when the robot fails to move any significant distance over an extended period of time. $\square$

Lemma 3. *If $\angle ABC < \frac{\pi}{2}$, then Algorithm 3 terminates with the robot at the circumcenter of ΔABC.*

Proof. Essentially, the robot moves in the direction opposite to the gradient of a bounded function $f(r) = d(\textsc{last}(r), r)$. This means that it will stop at the local minimum of f. If the triangle ΔABC is acute, then the only such minimum is the circumcenter (see Fig. 4). $\square$

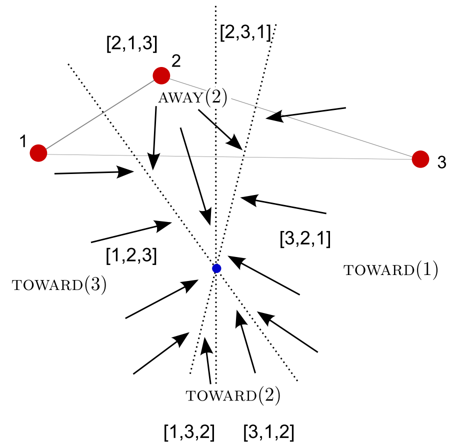


Fig. 5. Locating the circumcenter of an obtuse triangle. In some parts of bisector corresponding to BA and BC, direction fields' projections have opposite direction. According to Proposition 1, the final robot direction is aimed at the circumcenter.

Unfortunately, the previous algorithm does not work in case of an obtuse triangle, since now the local minima of f is not the circumcenter but the midpoint of AC. The following (less elegant) modification provides the solution.

Algorithm 4. *(Reaching the circumcenter of an obtuse triangle ΔABC).*

Description. First, the robot travels to the midpoint of AC. After having reached it, the robot continuously executes TOWARD(LAST(r)) unless B is the closest landmark; if it is, then the robot executes AWAY(B). Similarly to the previous algorithm, the robot stops when all three landmarks become (almost) equidistant. Fig. 5 provides the illustration. □

There are two potential issues with this algorithm. First, for some starting robot positions, AWAY(B) will direct the robot away from the landmarks and the circumcenter, and will never terminate. Beginning with the midpoint of AC deals with this, since it lies in the region in which the algorithm always terminates. Second, the use of the first part of Proposition 1 (opposite directions) is needed to determine the robot direction at some parts of the bisector lines, specifically the regions in which B switches to or from the first position in the distance ordering (see Fig. 5). However, it can be shown from geometric considerations that the robot movement is always directed toward the circumcenter.

Lemma 4. *Suppose $\angle ABC > \frac{\pi}{2}$. Algorithm 4 terminates with the robot at the circumcenter of ΔABC.*

Proof. See Appendix. □

Now we can derive the final algorithm for this section:

Algorithm 5. *(Reaching the circumcenter of any triangle ΔABC).*

Description. Run Algorithm 1 for each of the vertices of ΔABC to determine the type of the triangle. If all three angles are acute, then run Algorithm 3. If one of the angles is obtuse, relabel the landmarks so that the obtuse angle corresponds to vertex B and run Algorithm 4. If one of the angles is right, relabel the landmarks in the same way and run Algorithm 2. □

```
Input: $A, B, C \in L$
Output: TRUE if $\Delta ABC$ is Delaunay, FALSE otherwise
  $type \leftarrow acute$
  for all $Y$ in $\{A, B, C\}$ do
    $X, Z \leftarrow$ 2 remaining vertices
    TOWARD($X$)
    repeat
      TOWARD($Z$)
    until equal distance to $X$ and $Z$
    if $X$ is not closer than $Y$ then
      $(A, B, C) \leftarrow (X, Y, Z)$
      if $Y$ is closer than $X$ then
        $type \leftarrow obtuse$
      else
        $type \leftarrow right$
      break
  if $type = acute$ then
    repeat
      // LAST$_{ABC}(r)$ is the last of $A$, $B$, $C$ in the output
      TOWARD(LAST$_{ABC}(r)$)
    until equal distance to $A$, $B$, and $C$
  else if $type = obtuse$ then
    repeat
      if $B$ is closer than $A$ and $C$ then
        AWAY($B$)
      else
        TOWARD(LAST$_{ABC}(r)$)
    until equal distance to $A$, $B$, and $C$
  if FIRST($r$) $\in \{A, B, C\}$ then
    return TRUE
  else
    return FALSE
```

Fig. 6. Testing the Delaunay property

The correctness of the algorithm follows from Lemmas 1-4:

Theorem 1. *Algorithm 5 terminates with the robot at the circumcenter of ΔABC.*

C. Computing the Delaunay Triangulation

The Delaunay triangulation for a set of points in the plane is a triangulation such that no point is in the interior the circumcircle of any triangle [18]. We call a triangle ABC ($A, B, C \in L$) *Delaunay triangle* if it has the *empty circumcircle property*, i.e., its circumcircle contains no points of L.

Algorithm 6. *(Testing the Delaunay property of a triangle)*

Description. The robot executes Algorithm 5, while ignoring all the landmarks except A, B, and C. The triangle is Delaunay if and only if the three landmarks occur at the beginning of the distance ordering sensor output when the robot is at the circumcenter of ΔABC. See Fig. 6 for the pseudocode representation of the algorithm. □

Lemma 5. *Algorithm 6 is correct.*

Proof. The robot can safely ignore other landmarks in the sensor output for the first step of the algorithm, since the sensor reading for a subset of landmarks is exactly the corresponding subsequence of the full output.

Furthermore, by the general position assumption, no four landmarks lie on the same circle. Therefore a sensor output at the circumcenter of ΔABC that starts with A, B, C (in any order) implies that all other landmarks are outside of the circumcircle and the triangle is Delaunay. □

The faces of a Delaunay triangulation for a set of points are exactly all Delaunay triangles formed by those points [18]. Therefore, we can use the following algorithm to construct the Delaunay triangulation of L:

Algorithm 7. *(Computing the Delaunay triangulation)*

Description. Run Algorithm 6 for all triples of landmarks from L. If landmarks A, B, and C form a Delaunay triangle, then edges AB, AC, and BC belong to the Delaunay graph. □

IV. Using the Delaunay Triangulation

In this section we discuss certain tasks that can be performed using the information gathered in the previous section.

A. Computing the Convex Hull

The Delaunay triangulation of a set L encodes enough information to determine the convex hull of L. Indeed, the convex hull boundary is formed by the edges of a triangulation that belong to exactly one triangle. The only thing left is a trivial task of connecting edges in the correct order to form a closed path[2]. By using appropriate data structures (e.g., red-black trees), this algorithm can be adapted to run with $O(n \log n)$ time complexity, provided that the list of Delaunay triangles is precomputed in advance.

Algorithm 8. *(Computing the convex hull)*

Description. See Fig. 7 for the pseudocode implementation. □

```
Input: Set of landmarks $L$
Output: Convex hull of $L$
  $edges \leftarrow \varnothing$, $hull \leftarrow \varnothing$
  for all $A, B, C$ in $L$ do
    if IsDelaunay($\Delta ABC$) then
      for all $e$ in $\{AB, BC, AC\}$ do
        if not Contains($edges, e$) then
          Insert($edges, e$)
        else
          Delete($edges, e$)
  $e \leftarrow edges[0]$
  $v \leftarrow$ any endpoint of $e$
  Delete($edges, e$)
  Append($hull, v$)
  repeat
    $e \leftarrow$ element of $edges$ with endpoint $v$
    $v \leftarrow$ other endpoint of $e$
    Delete($edges, e$)
    Append($hull, v$)
  until IsEmpty($edges$)
  return $hull$
```

Fig. 7. Computing the convex hull

Suppose $M \subset L$ is a set of landmarks. The robot can construct the Delaunay triangulation of M by using the algorithms described in Section III and find the convex hull. However, it does not need to execute Algorithm 7 several times for different M. We can modify the algorithm to allow the robot to remember complete sensor outputs at every circumcenter of L. In this case, the robot would only need to run Algorithm 7 once for L in order to be able to compute the triangulation of any $M \subset L$ without moving anywhere. To do that, it can simply use the existing data and ignore all landmarks from $L \setminus M$.

[2]Videos demonstrating the execution of Algorithm 8 (implemented in simulation) are available at `http://msl.cs.uiuc.edu/~katsev1/publications/iros11/`

B. Patrolling

Similarly to [17], we define the task of patrolling a set of landmarks $M \subset L$, such that $M \cap \partial\text{hull}(L) = \emptyset$, as the problem of locating a *minimal* set $W \subset L$ such that $M \subset W$ and $M \cap \partial\text{hull}(W) = \emptyset$. The idea is that we want to find a route for the robot that would go around M, but not too far from it. In this case, the robot can patrol landmarks M by navigating the convex hull boundary of W. The following algorithm solves this problem.

Algorithm 9.

Description. Start with $W = L$. At every iteration try to find $l \in \partial\text{hull}(W)$ such that $M \cap \partial\text{hull}(W \setminus \{l\}) = \emptyset$ and replace W by $W \setminus \{l\}$. If no such l can be found, then W is minimal. □

It is unclear whether there exists an efficient algorithm to find the *smallest* set W that satisfies the requirements.

V. Conclusions And Future Work

We have analyzed the capabilities of the robot that is equipped with only one weak sensor that produces the distance ordering of the distinguishable landmarks. We have demonstrated that the robot can successfully reach the circumcenter of a triangle formed by any triple of landmarks. This enabled us to provide an algorithm for the robot to learn the Delaunay triangulation of the set of landmarks L. Using the information obtained, the robot is able to, for example, compute the convex hull of L or patrol a subset of L.

Many interesting questions remain for future research. The sensor output is based on the Euclidean distance ordering. Can it be replaced by a more general model? For example, can the underlying distance function be non-additive or lack radial symmetry? What other tasks can be performed by the robot using the information contained in the Delaunay triangulation? Although the Delaunay triangulation can be used for path planning, the robot can only compare length of intervals that have a common endpoint. Does this limitation make the optimal planning problem impossible? Algorithms of Section III operate only on three landmarks at time. Is there a better algorithm that will use all the landmarks simultaneously? How can obstacles incorporated in this problem (we might want to distinguish between two types of obstacles: physical obstacles that block robot movement and virtual obstacles that block sensing)?

Appendix

Proof of Lemma 4. Let $\textsc{first}(r)$ be the closest landmark, when the robot is at r, defined by

$$\textsc{first}(r) = \arg\min_{l \in L} d(r, l).$$

Consider the function

$$f(r) = d(r, \textsc{last}(r)) - d(r, \textsc{first}(r)),$$

which returns the difference between the distances to the closest and furthest landmarks. It is continuous by construction and its only minimum is the circumcenter of ΔABC, at which $f(r) = 0$.

Suppose that the robot is executing $\textsc{toward}(\textsc{last}(r))$ or $\textsc{away}(\textsc{first}(r))$ inside of a region in which the corresponding goal landmark does not change. In this case, f is non-increasing and even strictly decreasing, except for some locations on the extensions of triangle sides beyond the vertices. Indeed, when the robot moves directly toward $\textsc{last}(r)$, the distance to $\textsc{last}(r)$ decreases faster than the distance to any other landmark, including $\textsc{first}(r)$, therefore f is decreasing. The only exception is when the direction to $\textsc{last}(r)$ coincides with the direction to $\textsc{first}(r)$, in this case f stays constant. This requires the robot to be co-linear with $\textsc{last}(r)$ and $\textsc{first}(r)$ and not between them. It can be seen that it is not possible for this condition to continue indefinitely, therefore f will eventually decrease. Similar reasoning applies to the motion primitive $\textsc{away}(\textsc{first}(r))$.

We conclude that there are three possible scenarios for the robot: 1) it can move infinitely far away from the landmarks using the AWAY primitive; 2) it can get stuck, because the algorithm would instruct it to move in the opposite directions on two sides of the bisector line; 3) it can safely reach the minimum of f, the circumcenter of ΔABC.

Cases 1 and 2 only happen in certain locations. For example, 2 is impossible unless the robot is co-linear with B and A, or B and C. The algorithm's initial step is intended to drive the robot to the "safe" starting point, from which it is guaranteed to reach the circumcenter. □

Acknowledgments

This work is supported in part by NSF grant 0904501 (IIS Robotics), NSF grant 1035345 (CNS Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

References

[1] H. Bulata and M. Devy, "Incremental construction of a landmark-based and topological model of indoor environments by a mobile robot," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1054–1060, 1996.

[2] J.-B. Hayet, C. Esteves, M. Devy, and F. Lerasle, "Qualitative modeling of indoor environments from visual landmarks and range data," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and System*, pp. 631–636, IEEE, 2002.

[3] H. Hu and D. Gu, "Landmark-based navigation of mobile robots in manufacturing," in *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 121–128, 1999.

[4] M. Betke and L. Gurvits, "Mobile robot localization using landmarks," *IEEE Transactions on Robotics and Automation*, vol. 13, pp. 251–263, Apr. 1997.

[5] I. Shimshoni, "On mobile robot localization from landmark bearings," *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 971–976, Dec. 2002.

[6] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: a factored solution to the simultaneous localization and mapping problem," in *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 593–598, July 2002.

[7] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, and B. Stewart, "Distributed Multirobot Exploration and Mapping," *Proceedings of the IEEE*, vol. 94, pp. 1325–1339, July 2006.

[8] M. Erdmann and M. Mason, "An exploration of sensorless manipulation," *IEEE Journal on Robotics and Automation*, vol. 4, no. 4, pp. 369–379, 1988.

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

[10] M. Katsev, A. Yershova, B. Tovar, R. Ghrist, and S. M. LaValle, "Mapping and Pursuit-Evasion Strategies For a Simple Wall-Following Robot," *IEEE Transactions on Robotics*, vol. 27, pp. 113–128, Feb. 2011.

[11] E. Remolina and B. Kuipers, "Towards a general theory of topological maps," *Artificial Intelligence*, vol. 152, pp. 47–104, Jan. 2004.

[12] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Robotic exploration as graph construction," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 6, pp. 859–865, 1991.

[13] H. Choset and K. Nagatani, "Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 125–137, Apr. 2001.

[14] M. Blum and D. Kozen, "On the power of the compass (or, why mazes are easier to search than graphs)," in *19th Annual Symposium on Foundations of Computer Science*, pp. 132–142, IEEE, Oct. 1978.

[15] B. R. Donald, "On information invariants in robotics," *Artificial Intelligence*, vol. 72, pp. 217–304, Jan. 1995.

[16] M. Erdmann, "Understanding Action and Sensing by Designing Action-Based Sensors," *The International Journal of Robotics Research*, vol. 14, pp. 483–509, Oct. 1995.

[17] B. Tovar, L. Freda, and S. M. LaValle, "Learning Combinatorial Map Information from Permutations of Landmarks," *The International Journal of Robotics Research*, Oct. 2010.

[18] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[19] J. Chen, C. Luo, M. Krishnan, M. Paulik, and Y. Tang, "An enhanced dynamic Delaunay triangulation-based path planning algorithm for autonomous mobile robot navigation," in *Intelligent Robots and Computer Vision XXVII: Algorithms and Technique*, 2010.

[20] F. Gong and X. Wang, "Robot Path-Planning Based on Triangulation Tracing," in *International Symposium on Intelligent Information Technology Application*, pp. 713–716, Dec. 2008.

[21] M. Kallmann, "Path planning in triangulations," *IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, 2005.

[22] J. Jaromczyk and G. Toussaint, "Relative neighborhood graphs and their relatives," *Proceedings of the IEEE*, vol. 80, no. 9, pp. 1502–1517, 1992.

[23] F. Dehne, J. Sack, N. Santoro, J. Keil, and C. Gutwin, *The Delaunay triangulation closely approximates the complete Euclidean graph*, pp. 47–56. Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 1989.