The Raymond and
Beverly Sackler Faculty
of Exact Sciences
Tel Aviv University

The School of Computer Sciences

Localization with Single or Antipodal Distance Measurements

Thesis submitted in partial fulfillment of the requirements for the
M.Sc. degree in the School of Computer Sciences, Tel Aviv
University

By

Barak Ugav

The research work for the thesis has been carried out
under the supervision of
Prof. Dan Halperin

Acknowledgments

I would like to express my heartfelt gratitude to Dan Halperin, my direct adviser, for his guidance, expertise, and continuous support throughout the research process. Dan's mentorship and feedback have been instrumental in shaping the progression of my work and refining my ideas. It was a pleasure working with you, and I am grateful for the opportunity to learn from you.

I am also grateful to Efi Fogel for his exceptional coding assistance and technical expertise. Efi's contributions were vital in implementing and optimizing the practical aspects of my research.

Lastly, I am grateful to Steve Lavalle for his invaluable contributions to my research. Firstly, I would like to thank him for suggesting the research question that formed the foundation of my work. Secondly, I appreciate his collaborative approach, as he provided valuable ideas and suggestions throughout the research process. Steve's expertise and dedication have been instrumental in shaping the direction and methodology of my work, and I am honored to have had the opportunity to work alongside him.

Abstract

Given a polygon W, a *depth sensor* placed at point $p = (x, y)$ inside W and oriented in direction θ measures the distance $d = h(x, y, \theta)$ between p and the closest point on the boundary of W along a ray emanating from p in direction θ. We study the following problem: For a polygon W with n vertices, possibly with holes, preprocess it such that given a query real value $d > 0$, one can efficiently compute the preimage $h^{-1}(d) \subset W \times \mathbb{S}^1$, namely determine all the possible poses (positions and orientations) of a depth sensor placed in W that would yield the reading d, in an output-sensitive fashion. We describe such a data structure requiring $O((E+n)\log n)$ preprocessing time, $O(E)$ storage space and $O(1+k\log n)$ query time, where E is the number of edges in the *visibility graph* of W (the graph whose nodes are the vertices of W, and a pair of nodes is connected by an edge if and only if the open line segment connecting the corresponding vertices is fully contained in W), and k is the number of vertices and maximal arcs of low degree algebraic curves constituting the answer. We also obtain analogous results for the more useful case (narrowing down the set of possible poses), where the sensor performs two antipodal depth measurements from the same point in W. The query time in the latter structure is $O(\log n + k)$, while the preprocessing time and storage are the same as in the former structure. We then describe simpler data structures for the same two problems, where we employ a decomposition of $W \times \mathbb{S}^1$, which is an extension of the well-known trapezoidal decomposition, and which we refer to as *rotational trapezoidal decomposition*. The answers to queries in these data structures are output-sensitive relative to this decomposition: If k cells of the decomposition contribute to the final result, we will report them in $O(1+k)$ (resp., $(\log n + k)$) time for one measurement (resp., antipodal measurements), after $O((E+n)\log n)$ preprocessing time and using $O(E)$ storage space. These problems are inspired by *localization* questions in robotics, where we are given a map of the environment, a robot is placed in an unknown pose, and we wish to determine where the robot is. Although localization is often carried out by exploring the full *visibility polygon* of a sensor placed at a fixed point of the environment, the approach that we propose here opens the door to sufficing with only few depth measurements, which is advantageous as it allows for usage of inexpensive sensors and could also lead to savings in storage and communication costs.

Contents

Chapter 1

Introduction

A depth sensor that provides a single distance measurement[1] is placed inside a known polygonal workspace $W \subset \mathbb{R}^2$ with n vertices. Consider the depth mapping $d = h(x, y, \theta)$ in which $(x, y) \in W$ is the position of the sensor and $\theta \in \mathbb{S}^1$ (namely $[0, 2\pi)$) is the direction of the ray emanating from the sensor and along which the distance to the workspace boundary is measured. The set of all sensor poses (x, y, θ) forms a three-dimensional configuration space (C-space for short). We wish to devise a data structure such that, after preprocessing the workspace, will efficiently answer the following queries: Given a distance measurement (real number) $d > 0$, report all the possible sensor configurations that could yield such a measurement. In other words, we aim to compute h's preimage $h^{-1}(d) = \{(x, y, \theta) \in W \times \mathbb{S}^1 \mid d = h(x, y, \theta)\}$.

Determining a sensor's location is one of the main tasks of sensor fusion systems. Examples are ubiquitous, including Global Positioning Systems (GPS), celestial navigation, trilateration, camera pose estimation, and localization of autonomous vehicles. Our problem is motivated mostly by the last one, in which a mobile robot has a complete map of its environment and must use sensing to determine its *pose* (position and orientation) with respect to the map. This so-called *robot localization* problem is crucial to robotics and has been researched for several decades.

1.1   Related Work

Many variants of the robot-localization problem were studied throughout the years, both where the environment is known and the robot should localize itself in it, or where the environment is not known in advance and the robot should simultaneously learn and map the environment, and localize itself.

The simultaneous localization and mapping (SLAM) variant, in which the environment is not known, is harder in the sense that the robot has less information than the variant where the environment is known, and therefore methods aiming to solve it often yield im-

---

[1]Throughout the thesis we use *depth measurement* and *distance measurement* interchangeably.

precise results (see, for example, [12, 30]). The robot is required to move and probe the environment using its sensor(s) (such as a camera, a distance probe, etc.) multiple times, acquiring sufficient information to identify its surrounding. Earlier work used find-grained grids (typically square or hexagonal grids) to discretize the environment and represent free or occupied regions [13, 14]. A common alternative to the grid discretization, is to represent the map as a collection of sparse features extracted from the sensors data stream [10]. Since the 1990s, a variety of probabilistic techniques dominate the field [33]. The SLAM approach is unavoidably affected by a great deal of uncertainty, and these probabilistic techniques are crucial in order to be able to overcome this challenge, as they do not commit to possibly noisy observations, but rather use probability distributions. Sensors commonly used are GPS, compasses, lasers and radars, which are essential for gathering sufficient data about the environment where the robot moves, and while they provide the information required to understand the robot position, all of them yield measurements with errors in real world use, and the probabilistic techniques deal with the noise of these measurements. While the robot moves and samples measurements using its sensors, an estimation of the environment and the current position are maintained, using a distribution over the range of possible values. Examples of such commonly used probabilistic techniques are SLAM solutions using (Extended) Kalman Filter [10, 21, 24, 29], which is an algorithm that estimates the state of the environment and the robot given a series of measurements observed over time, including statistical noise and other inaccuracies. The classical Kalman Filter assumes that the dynamics of the system are linear, while the Extended Kalman Filter is a nonlinear version of the algorithm.

As part of a SLAM algorithm, a subproblem arises, of *incremental* localization (also called *relative*, or *position/pose tracking*), in which the configuration estimate was known a moment ago and must be updated based on recent motions and sensor readings; see for example [26]. In case the robot already mapped the environment, but experiences failures and looses its current pose, another subproblem is encuntered, the so-called *kidnapped robot problem* [15] (also called *absolute*, *relocation* or *global localization*), in which the robot must determine its configuration "from scratch". In the latter variant, the environment is assumed to be known and we are only required to identify where the robot is within it. The kidnapped-robot version resembles our problem and could correspond to such a robot using SLAM, or to a freshly deployed or rebooted robot that has a description of the environment in the first place.

Previous research on the kidnapped-robot problem took different approaches, such as discretizing the environment into grid-like cells [5, 6]. Then, the robot should be located in one of these cells, for example by maintaining a real value for each cell that represent the probability the robot is located in that cell and updating these values when further measurements are available. A slightly different approach is to use sample random points, instead of using fixed grids, and estimating for each such random point if it is the current position of the robot [9].

There are both an *active* variant [11, 23, 25, 27], in which the robot must determine where to move next to quickly reduce uncertainty, and a *passive* variant, in which the sensor fusion system works with whatever sensing and motion data are available. Passive approaches

mainly address stochastic uncertainties [17] and are typically integrated with mapping to obtain SLAM [10, 32].

This thesis considers the passive localization case and stands out from previous work in that it uses much less sensor data. Most robot localization methods are based on imaging sensors, such as cameras or Lidars, which provide a dense collection of measurements [33]. From a computational geometry viewpoint, these may be considered as providing visibility polygons (see, e.g., [1, 4, 20, 34]), whereas our sensor analogously provides only a single visibility ray with each measurement. Localization with only few depth measurements is advantageous over exploring the full visibility polygon in that it allows for using inexpensive sensors and requires less storage space and communication. Our current work, which focuses on one or two measurements, is a step forward in this direction. However, it still falls short of giving a full localization answer, which in general requires at least three measurements; see further discussion of this issue in Section 7.

1.2   Contribution and Outline

Let W be a polygon in the plane, possibly with holes, and having n vertices. The *visibility graph* of W is the graph whose nodes are the vertices of W, and a pair of nodes is connected by an edge if and only if the open line segment connecting the corresponding vertices is fully contained in W. The contributions described in the thesis are as follows:

(i) We review in Chapter 2 an extension of the planar trapezoidal decomposition (see, e.g., [8, Chapter 6]) to the three-dimensional space $W \times \mathbb{S}^1$. The decomposition, which we refer to as *rotational trapezoidal decomposition* (RTD, for short), has $O(E)$ cells of constant complexity each, and it can be computed in $O((E + n) \log n)$ time, where E is the number of edges in the visibility graph of W, which is bounded by $O(n^2)$ in the worst case. The RTD is easier to construct than the standard three-dimensional vertical decomposition [7, 22] and it suits the problem at hand better, as the imaginary walls of the RTD are parallel to the measurement direction.

(ii) In Chapter 3 we use this decomposition to build a simple data structure to answer the preimage (see Figure 1.1) queries—given a real value $d \geqslant 0$, determine $h^{-1}(d)$—in $O(1 + k)$ time, where k is the number of cells of the RTD that contain part of the answer. The preprocessing time remains $O((E + n) \log n)$ and the required storage is $O(E)$—this remains true for the following data structures as well. We analyze (Section 3.2) the precise shape of the projection of the preimages $h^{-1}(d)$ onto the workspace. These projections could also serve as an answer to a measurement query as they delineate the regions in the plane (positions only) where the sensor could be.

(iii) The data structure of contribution (ii) is output sensitive with respect to the (artificial) RTD. One of **the main contribution of the thesis** consists of a more elaborate data structure that is output-sensitive in the sense that the query time is proportional, up to a logarithmic factor, to the number k of vertices and maximal arcs of low-degree algebraic curves that constitute the output. In Chapter 4 we describe such data structure for the one-measurement case that answer a query in $O(1 + k \log n)$.

(iv) In Chapter 5 we discuss the following two-measurement query type: Given two antipo-

dal depth measurements of a sensor from the same (unknown) point in W in the (unknown) directions θ and $\theta + \pi$, determine all the poses of the sensor that would yield these measurements. In Section 5.1 we devise a data structure that answers the antipodal queries in $O(\log n + k)$ time where k is the number of cells of the RTD that contain part of the answer. Similar to contribution (ii), the RTD-based data structure for antipodal measurements is output sensitive with respect to the RTD, and in Section 5.3 we present one of **the main contribution of the thesis** which is an output-sensitive data structure for antipodal measurements that answer a query in $O(\log n + k)$ time where k is the number of vertices and maximal arcs of low-degree algebraic curves that constitute the output.

(v) We implemented the RTD-based data structures, and we demonstrate a few results obtained with the implementation. The code is open source and is available for download; see Chapter 6.

Throughout the thesis, we assume general position of the input: no three vertices lie on the same line, no three edges intersect at the same point, no two edges are parallel, etc. (We give up on the general position assumption in Figure 1.1 for simplicity of demonstration.)
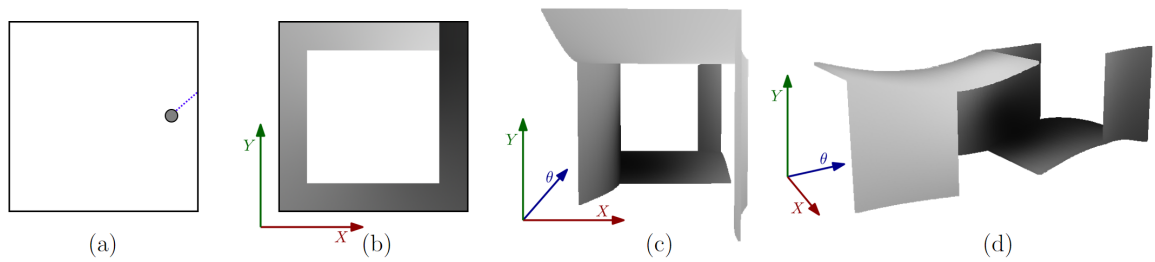

Figure 1.1: Given a sensor drawn as a gray disc in a unit square workspace (a), and a single distance measurement $d = 0.15$ drawn as a dotted line sourced at the sensor and hitting the wall, the possible locations and configurations of the sensor are shown in the two-dimensional workspace (b), and in the three-dimensional C-space (c,d), respectively. The gray scale corresponds to the measurement angle: configurations with greater measurement angle are drawn in darker gray. The gray shades are projected onto the two-dimensional workspace at (b); note that, as there are multiple configurations with different measurement angles that are projected onto the same position p, the shade at p is the darkest shade among all of them.

Chapter 2

Rotational Trapezoidal Decomposition

We will describe a few data structures that are built upon a known decomposition of the three-dimensional C-space $\mathbb{R}^2 \times \mathbb{S}^1$, which we refer to as *rotational trapezoidal decomposition*, or RTD for short. This decomposition has been previously used in other problems involving three-dimensional configurations spaces involving translation and rotation in the plane; see, e.g., [28]. Conceptually, the decomposition generalizes the known, planar, vertical decomposition [8, Chapter 6] to any direction. The planar trapezoidal decomposition partitions the workspace W to a set of trapezoids.[1] A trapezoid is identified by its ceiling and floor (namely top and bottom edges, respectively), and the left and right vertices, which define the left and right artificial vertical edges. See Figure 3.1 for an illustration. We remark that there are other possible decompositions of *arrangements* of surfaces in three-dimensional space [22, Section 28.3], but RTD seems most suitable for our purpose.

We can rotate the scene to any orientation and compute all the trapezoids of any angle. As we let the orientation θ of the decomposition vary a little, we notice that the trapezoids remain similar (in a sense to be made precise shortly) unless they undergo some combinatorial change. As the direction of decomposition changes, the trapezoid changes continuously, and we refer to the union of these trapezoids as a three-dimensional cell, or *cell* for short. Since throughout the change of orientation the ceiling and floor edges, as well as the vertices determining the left and right boundaries of the trapezoid remain the same we use the following notation: A cell C is identified by its ceiling (top) and floor (bottom) edges, e_t^C, e_b^C, and the left and right vertices, v_l^C, v_r^C, which define the left and right artificial edges (parallel to the direction of the decomposition). The two limiting vertices, v_l^C, v_r^C, may be endpoints of e_t^C, e_b^C, or other reflex vertices of the workspace.

Now, in addition to describing a cell by its floor, ceiling and the two vertices that define the two artificial side edges, we also add the first and last orientations where the trapezoid exists (in the combinatorial sense, namely having the same edges and vertices defining it). These cells constitute a decomposition of the three-dimensional C-space of the sensor. With this description, there are at most $O(n^2)$ unique cells, and we can compute each cell's exact boundary as a function of the orientation θ of the rotated workspace. More precisely, there

---

[1]Some of the two-dimensional cells in the decomposition may be triangles; for brevity we will also refer to them in this context as trapezoids.

are $O(E)$ such unique cells, where E is the number of edges in the visibility graph of W, which is bounded by $O(n^2)$.

2.1 Computing the Rotational Trapezoidal Decomposition

We construct the cells of the RTD by simultaneously performing a radial sweep around all the vertices of the workspace, where each vertex is the origin of a sweeping ray, all rays point at the same direction and are rotated together. An event occurs when one of the rays originating at some vertex u hits another vertex v that is visible from u. We compute all the events in advance by computing the visibility graph (VG) of W in $O(E + n \log n)$ time using the algorithm of Ghosh and Mount [18] and creating for each edge (u, v) in the VG two events (one originating at u with direction $\overrightarrow{uv}$ and the other originating at v with the opposite direction), and then radially sorting all of them by θ. In addition, each ray stores at all times (angles) the cells on both its sides, allowing us to keep track of all the existing trapezoids of the decomposition at the current orientation. During the handling of an event, where two vertices align along the sweeping ray (its origin and another vertex), we know they are visible from one another (as the events were created from the visibility graph), namely the open line segment connecting them lies in the interior of the workspace, and cells should be created or terminated. There are two types of events along the radial sweep:

Type I Two vertices of the same edge are incident to a rotating ray. See Figure 2.1 for an illustration. The triangular cell that contained both vertices on its boundary is "squeezed" and terminated, and a new triangular cell is created. The cell sharing an artificial edge with the triangular cell also terminates and a new one is created due to change of one of its limiting vertices. In total, two cells are terminated and two are created.

Type II Two vertices of two different edges are incident to a rotating ray. See Figure 2.2 for an illustration. The middle cell is "squeezed" and terminated, and a new middle cell is created. The cells from both sides of the connecting ray terminate and new ones are created due to change in the limiting vertices. In total, three cells are terminated and three are newly created.

In both event types, the top and bottom edges and left and right vertices are known locally from the information maintained by the rays or the terminated cells. During handling of an event we terminate or create a constant number of cells. As there are $O(E)$ events, there are also $O(E)$ cells in total.

To start the process, trapezoidal decomposition in the horizontal direction (namely, with $\theta = 0$), is performed, requiring $O(n \log n)$ time and starting $\Theta(n)$ cells. We let θ vary in the range $[0, 2\pi)$. Notice that the trapezoidal decomposition at $\theta = 0$ artificially cuts cells into two; this however does not affect the analysis that follows or the asymptotic resources required by the algorithm. In total, we construct these $O(E)$ cells and their angle intervals
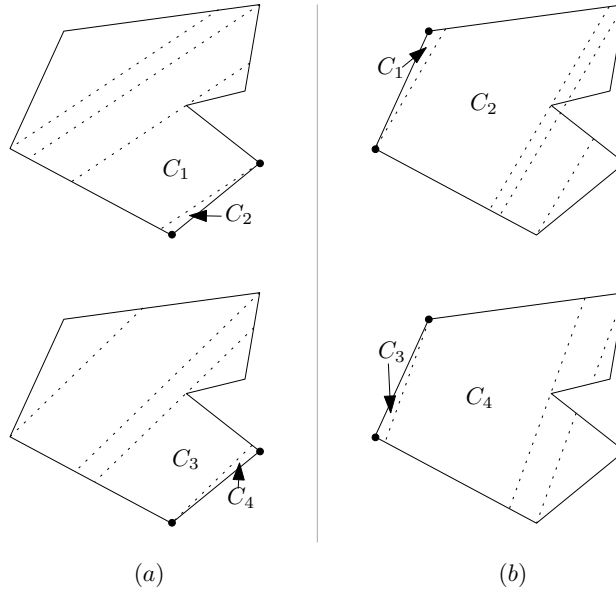
Figure 2.1: Type I events: Two vertices of the same edge align along the rotating ray. Rays are rotating counterclockwise. The event vertices are drawn as small black discs. Each column depicts one example, before (top) and after (bottom) the event.

in $O((E + n)) \log n)$ time, while using $O(E + n)$ space. It is easy to see that $\Omega(n^2)$ cells may be created and so the bound on the maximum number of cells is tight. The process is summarized in the following lemma.

Lemma 2.1.1. *Given a polygonal workspace W with a total of n vertices, the three-dimensional configuration space $W \times \mathbb{S}^1$ is partitioned by the Rotational Trapezoidal Decomposition (RTD) described above into $O(E)$ cells, each of constant descriptive complexity, where E is the number of edges in the visibility graph of W. The construction of the decomposition takes $O((E + n) \log n)$ time and requires $O(E + n)$ storage space. The number of edges E can be $\Theta(n^2)$ in the worst case.*

2.2 The RTD Opening Function

Each cell C prevails through an interval of θ values, calculated during the sweep—we denote this interval by $\Theta^C = [\theta^C_{begin}, \theta^C_{end})$. Given a cell C and an angle $\theta \in \Theta^C$, we can easily construct the trapezoid that is the cross-section of C at θ. The top left and top right vertices (which are not necessarily vertices of the workspace) of the trapezoid both lie on e^C_t. We can compute their x values as a function of θ, denoted $x^C_{tl}(\theta), x^C_{tr}(\theta)$. Assume, w.l.o.g., that the top and bottom edges are not vertical, and let $y = e^C_t(x)$ be the function that maps an x-coordinate to a y value of a point (x, y) on e^C_t, and similarly $y = e^C_b(x)$ for the bottom edge.

Definition 2.2.1. *The opening of a cell C at angle θ and at a point $(x, e^C_t(x))$ on the top edge of the cell, is the length of the intersection of the line with slope θ through the point*
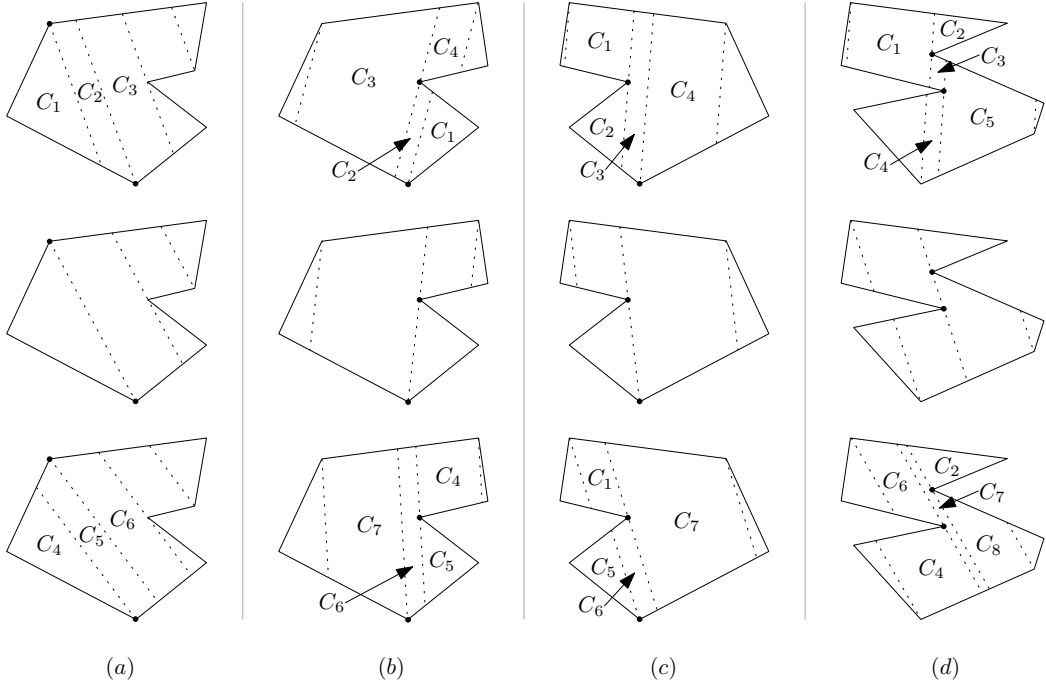
7

Figure 2.2: Type II events: Two vertices of two distinct edges align along the rotating ray. Rays are rotating counterclockwise. The event vertices are drawn as small black discs. Each column depicts one example, before (top), during (middle), and after (bottom) the event.

$(x, e_t^C(x))$ *on the top edge of the cell, with the trapezoidal cross-section of C at angle θ. We denote this opening by $O^C(\theta, x)$.*

Next, we show a few properties of the opening function:

- The opening function can be analytically described as a function of θ and x: Given a cell $C(e_t^C, e_b^C, v_l^C, v_r^C, \theta^C)$, assume w.l.o.g that the top and bottom edges are not vertical, let $e_t^C(x) = x m_{e_t} + b_{e_t}$ be the equation for the top edge and analogously for the bottom edge $e_b^C(x) = x m_{e_b} + b_{e_b}$.

  For any x, denote by $\bar{z}$ the vertical segment at x with one endpoint on e_b^C and the other endpoint on e_t^C, and denote its length by $|\bar{z}|$. Consider the triangle whose edges are O, $\bar{z}$ and a portion of e_b^C (see Figure 2.4 for an illustration). The angle opposite e_b^C is $\frac{\pi}{2} - \theta$, the angle opposite of $\bar{z}$ is $\theta - \gamma$ and the angle opposite O is $\frac{\pi}{2} + \gamma$. Using the law of sines:

$$O(\theta, x) = \frac{|\bar{z}| \sin(\frac{\pi}{2} + \gamma)}{\sin(\theta - \gamma)} = \frac{\cos \gamma}{\sin(\theta - \gamma)} = \frac{\cos \gamma}{\sin \theta \cos \gamma - \cos \theta \sin \gamma}$$

$$= \frac{\frac{1}{\sqrt{1 + m_{e_b}^2}}}{\sin \theta \frac{1}{\sqrt{1 + m_{e_b}^2}} - \frac{m_{e_b}}{\sqrt{1 + m_{e_b}^2}} \cos \theta} = \frac{1}{\sin \theta - m_{e_b} \cos \theta} = \frac{e_t^C(x) - e_b^C(x)}{\sin \theta - m_{e_b} \cos \theta}$$

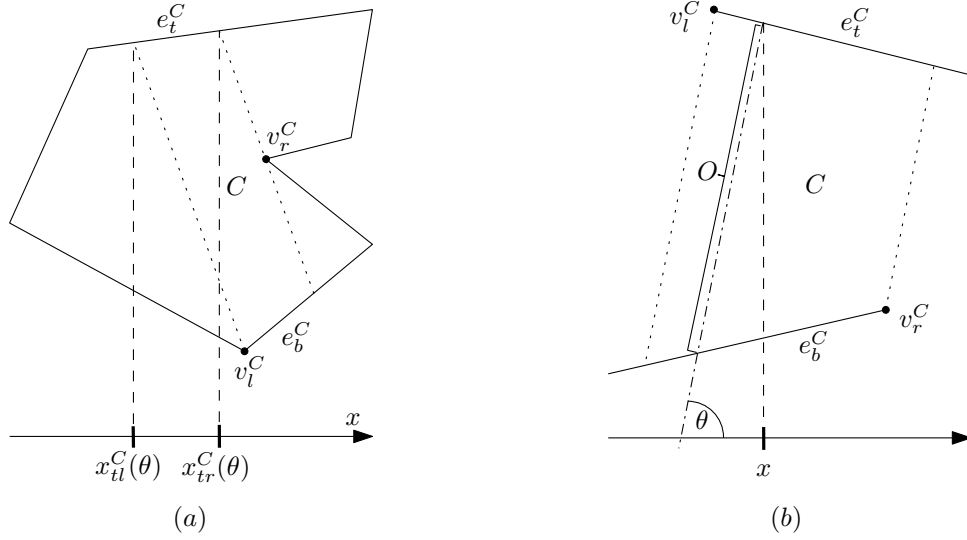$$= \frac{x(m_{e_t} - m_{e_b}) + b_{e_t} - b_{e_b}}{\sin \theta - m_{e_b} \cos \theta} .$$

8

Figure 2.3: (a) The x values of the left and the right vertices of a cell's ceiling. (b) The opening size O as a function of x and θ.
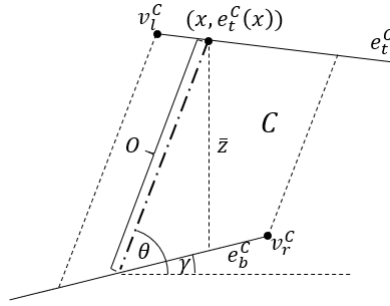

Figure 2.4: The opening function of a single cell, for fixed values of x and θ, is drawn along with the vertical segment $\bar{z}$ at x.

- The inverse of the opening function can be analytically described: For each RTD cell C, solving $O(\theta, x) = f$ for a specific value f is required by later data structures. Given a cell $C(e_t^C, e_b^C, v_l^C, v_r^C, \theta^C)$, assume w.l.o.g that the top and bottom edges are not vertical, and denote by $e_t^C(x) = x m_{e_t} + b_{e_t}$ the equation for the top edge and equivalently for the bottom edge $e_b^C(x) = x m_{e_b} + b_{e_b}$. Using the opening equation:

$$O^C(\theta, x) = f \ ,$$

$$\frac{e_t^C(x) - e_b^C(x)}{\sin\theta - m_{e_b}\cos\theta} = f \ ,$$

$$e_t^C(x) - e_b^C(x)) = f(\sin\theta - m_{e_b}\cos\theta) \ ,$$

$$m_{e_t}x + b_{e_t} - m_{e_b}x - b_{e_b} = f(\sin\theta - m_{e_b}\cos\theta) \ ,$$

$$x = \frac{f(\sin\theta - m_{e_b}\cos\theta) + b_{e_b} - b_{e_t}}{m_{e_t} - m_{e_b}} \ .$$

9

2.3 The Vertices of a Trapezoid

For each RTD cell C, which exists in an angle interval Θ^C, an exact trapezoid description can be calculated analytically for any fixed $\theta \in \Theta^C$. In this section we give an exact formula for the two top vertices of a trapezoid, $x_{tl}^C(\theta), x_{tr}^C(\theta)$.

The top and bottom edges are fixed, and the two artificial edges are oriented at angle θ. It remains to express the vertices created by the intersections of the artificial edges and the top edge, denoted v_{tl}, v_{tr} with x values $x_{tl}^C(\theta), x_{tr}^C(\theta)$. Given a cell $C(e_t^C, e_b^C, v_l^C, v_r^C, \theta^C)$, assume, w.l.o.g., that the top and bottom edges are not vertical, and denote by $e_t^C(x) = x m_{e_t} + b_{e_t}$ the equation for the top edge and similarly for the bottom edge $e_b^C(x) = x m_{e_b} + b_{e_b}$. Let $(x_{v_{tl}}, y_{v_{tl}})$ be the coordinates of the left vertex v_l^C defining the trapezoid, then any point (x, y) on the left artificial edge can be expressed as (see Figure 2.3a):

$$y = \tan\theta(x - x_{v_{tl}}) + y_{v_{tl}} \ .$$

To calculate the intersection with the top edge, we substitute the top edge equation:

$$m_{e_t} x + b_{e_t} = x \tan\theta - x_{v_{tl}} \tan\theta + y_{v_{tl}} \ ,$$

$$x(m_{e_t} - \tan\theta) = y_{v_{tl}} - x_{v_{tl}} \tan\theta - b_{e_t} \ ,$$

$$x = \frac{y_{v_{tl}} - x_{v_{tl}} \tan\theta - b_{e_t}}{m_{e_t} - \tan\theta} \ .$$

Similar calculation gives the right top vertex. In conclusion:

$$x_{tl}^C(\theta) = \frac{y_{v_l} - x_{v_l} \tan\theta - b_{e_t}}{m_{e_t} - \tan\theta} \ , \quad x_{tr}^C(\theta) = \frac{y_{v_r} - x_{v_r} \tan\theta - b_{e_t}}{m_{e_t} - \tan\theta} \ .$$

Chapter 3

RTD-Based Single Distance Measurement Localization

In this chapter we describe a data structure to determine the possible configurations of a sensor given a single distance measurement d taken from an unknown position in an unknown direction. The data structure uses RTD, the decomposition described in Chapter 2, and achieves a query time of $O(1 + k)$, where k is the number of RTD cells that contains part of the result. Note that such query time is optimal with respect to the RTD, but fails to achieve real output sensitivity—this requires a more involved data structure, which we will present in the next chapter, Chapter 4.

3.1   RTD-Based Data Structure Construction

We use the decomposition described in Chapter 2 of the three-dimensional C-space of the sensor into cells of constant descriptive complexity each, such that within each cell C, the sub-region $C_d = \{(x, y, \theta) \in C \mid d = h(x, y, \theta)\}$ has constant descriptive complexity as well. We then show, how to maintain the cells of the decomposition in a search structure, such that given a query d we can easily retrieve all the cells C for which $C_d \neq \varnothing$, in time proportional to their number.

Suppose for the purpose of exposition that we know that the sensor measures its distance in the upward vertical direction, namely $\theta = \pi/2$. We apply trapezoidal (vertical) decomposition to the workspace. Given a measurement d, we can easily compute for each trapezoid in the decomposition whether there are any points in it that are at vertical distance d from the top edge of the trapezoid. These points lie along a segment parallel to the top edge, and the desired answer is the union all such segments over all trapezoids. See Figure 3.1 for an illustration.

The direction of trapezoidal decomposition is parallel to the direction of measurement, therefore for us to remove the assumption that the sensor measures its distance in the vertical direction, instead of considering the result segments within each trapezoid of the vertical
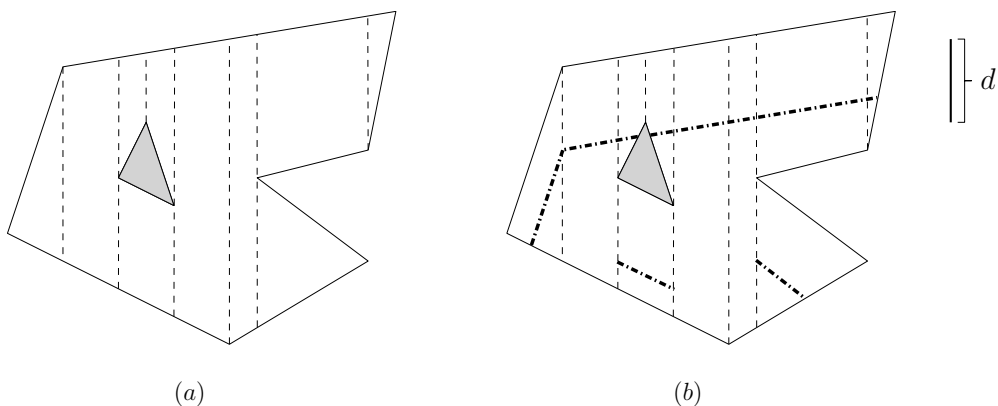
Figure 3.1: Example of a vertical decomposition of a polygonal workspace with a single hole (a) and the line segments (bold, dash-dotted) along which the sensor may be when it reads the distance d in the upward vertical direction (b).

decomposition, we consider the sub-region C_d within each cell C in the RTD decomposition. Each such cell has constant complexity and the result sub-region is easy to compute.

We wish to store the cells of the decomposition of the C-space of the sensor in a data structure such that given a distance measurement d, we can efficiently retrieve all the cells that have a non-empty set of candidate poses for this measurement. To this end we define the maximal opening O_{max}^C of a cell C, which is the maximal-length θ-oriented segment that can be placed in a θ-cross-section of a cell C, over all possible values $\theta \in \Theta^C$.

We introduced the opening function in the RTD section (Definition 2.2.1). Now we define for a cell C the *maximal opening*, which is the maximum value of the opening function for any valid values of θ, x:

$$O_{max}^C = \max_{\theta \in \Theta^C} \max_{x \in [x_{tl}^C(\theta), x_{tr}^C(\theta)]} O^C(\theta, x) \ .$$

In a preprocessing phase, we calculate O_{max}^C for each cell, and keep the cells sorted in decreasing order by this value in an array. Given a query measurement d, the cells that contain potential poses for this reading are exactly the cells for which $d \leqslant O_{max}^C$. We can find these cells by traversing the sorted cells in descending maximum opening order, resulting in $O(1 + k)$ query time,[1] where k is the number of cells that contain workspace points of the answer. The calculation of the maximum opening takes constant time per cell. Since there are $O(E)$ cells overall, the entire preprocessing requires $O((E + n) \log n)$ time and $O(E + n)$ space, which is asymptotically subsumed by the construction of the decomposition (Chapter 2).

Theorem 3.1.1. *Given a polygonal workspace with a total of n vertices, the RTD cells of the configuration space $W \times \mathbb{S}^1$ can be computed and sorted in decreasing order of their maximum opening in time $O((E + n) \log n)$ and $O(E + n)$ space, where E is the number of edges in the visibility graph of W. Then, given a query real parameter $d > 0$, we can report all the cells*

---

[1]For $k = 0$ we still need to check the cell at the top of the list to find out that no relevant cells exist.

that contain a possible configuration yielding depth measurement d in time $O(1 + k)$, where k is the number of the relevant cells.

This structure has a couple of advantages: (i) it is trivial to construct and implement (see Chapter 6), and (ii) it is worst-case optimal in storage space and query time *assuming the Rotational Trapezoidal Decomposition.* However, RTD subdivides the C-space in a way that may induce large unnecessary overhead for certain queries (see Section 3.3). In Chapter 4 we provide an improved solution with only a logarithmic factor in query time over the minimum possible output size.

3.2   Sensor Positions in a Single RTD Cell

Given a single sensor reading d, we concern ourselves in this section with the shape and complexity of the locus of sensor *positions* that result in such a reading, namely ignoring the orientation of the sensor's ray. We believe that reporting the possible positions only is, although incomplete, useful feedback that is more intuitive and easy to grasp by the user of a robot system. This in itself can serve as an answer to a query, in particular, when we wish to further process the regions using additional information about the potential sensor pose.

For each three-dimensional cell C of the RTD, let $\Psi_d(C)$ denote the union of positions where the sensor could be, namely $\Psi_d(C)$ is the projection onto the xy-plane of C_d. Fix a cell C with top and bottom edges e_t^C, e_b^C, limiting vertices v_l^C, v_r^C, which exist in an angle interval Θ^C. Recall that $x_{tl}^C(\theta), x_{tr}^C(\theta)$ denote the x value of the left and right top vertices of the trapezoidal cross-section of C (which are not necessarily vertices of the workspace), respectively, at angle θ.

Let $p_l^C(\theta)$ denote the position of the sensor, where a distance measure d at angle θ will hit e_t^C at x value $x_{tl}^C(\theta)$, while ignoring other obstacles in the workspace. Specifically, the point whose x-coordinate is $x_{tl}^C(\theta) - d\cos\theta$ and y-coordinate is $e_t^C(x_{tl}^C(\theta)) - d\sin\theta$, where $e_t^C(\cdot)$ is the equation of the line supporting the top edge. Similarly denote by $p_r^C(\theta)$ the sensor pose that will result in hitting e_t^C at $x_{tr}^C(\theta)$. Recall that $\Psi_d(C)$ denotes the region $\{(x,y)|(x,y,\theta) \in C \text{ and } h(x,y,\theta) = d\}$, namely the positions of the sensor in the workspace, where the sensor configuration is in C and there is a sensor reading d. We wish to trace the curves drawn by $p_l^C(\theta)$ and $p_r^C(\theta)$ as θ varies in the range Θ^C. These curves are portions of the boundary of the region $\Psi_d(C)$. The type of the curve that the function $p_l^C(\theta)$ (respectively, $p_r^C(\theta)$) draws, depends on v_l^C (respectively, v_r^C), the vertex defining the left (respectively, right) wall of the cell. We describe it here for v_l^C. The description for v_r^C is analogous.

Type A The vertex v_l^C defining the left wall of the trapezoid/cell lies on e_t^C. See Figure 3.2(a) for an illustration. The top left vertex, and therefore $x_{tl}^C(\theta)$, are fixed (i.e., independent of θ), and the curve is a circular arc of radius d around v_l^C: $v_l^C - (d\cos\theta, d\sin\theta)$.

Type B The vertex v_l^C defining the left wall of the trapezoid/cell does *not* lie on e_t^C. See Figure 3.2(b,c) for illustrations. The curve traced by $p_l^C(\theta)$ is a conchoid of Nicomedes;
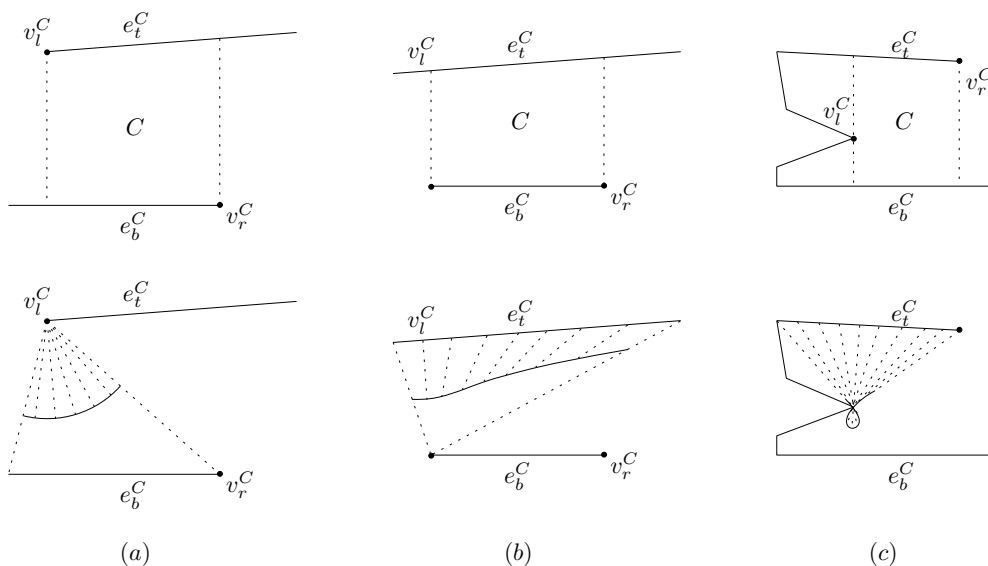
see Appendix A.1 for details.

Figure 3.2: (a) Type A curve: when the limiting vertex lies on the cell's ceiling, the curve is a circular arc. (b)(c) Type B curve: when the limiting vertex does not lie on the cell's ceiling, the curve is a conchoid of Nicomedes.

For any angle $\theta \in \Theta^C$, all points on the line segment $(p_l^C(\theta), p_r^C(\theta))$ are points the robot can be at and measure d at e_t^C (ignoring other obstacles). Together with the types of the curves traced by $p_l^C(\theta), p_r^C(\theta)$, we have an exact description of the region of the workspace containing all the points the robot might be at, while its configuration is in the cell C.

If the interval Θ^C contains the angle perpendicular to the top edge, denoted θ_p^C, we divide it into two sub-intervals, one with angles that are greater than or equal to the perpendicular angle and the other with smaller angles, $[\theta_{begin}^C, \theta_p^C), [\theta_p^C, \theta_{end}^C)$. A single shape per sub-interval will be created. If θ_p^C is not included in Θ^C, a single interval $[\theta_{begin}^C, \theta_{end}^C)$ and a single corresponding shape is used.

We handle each of the two sub-intervals separately. For each sub-interval (or for the entire interval Θ^C when no splitting is required) $[\theta_1, \theta_2)$, we describe all the points on the line segments $(p_l^C(\theta), p_r^C(\theta))$ for any $\theta \in [\theta_1, \theta_2)$ by a shape defined by the vertices $(v_1, v_2, v_3, v_4) = (p_l^C(\theta_1), p_l^C(\theta_2), p_r^C(\theta_2), p_r^C(\theta_1))$ and the edges p_l^C, p_r^C between $(v_1, v_2), (v_3, v_4)$ respectively and straight line edges between $(v_2, v_3), (v_4, v_1)$. The union of the two shapes is the desired two-dimensional region in the workspace the sensor might be in. The only obstacle we need to consider is the bottom edge e_b^C of the trapezoid, and we can simply intersect the result with the upper half-plane defined by the line through e_b^C to get the final result of the cell. See Figure 3.3 for an illustration.
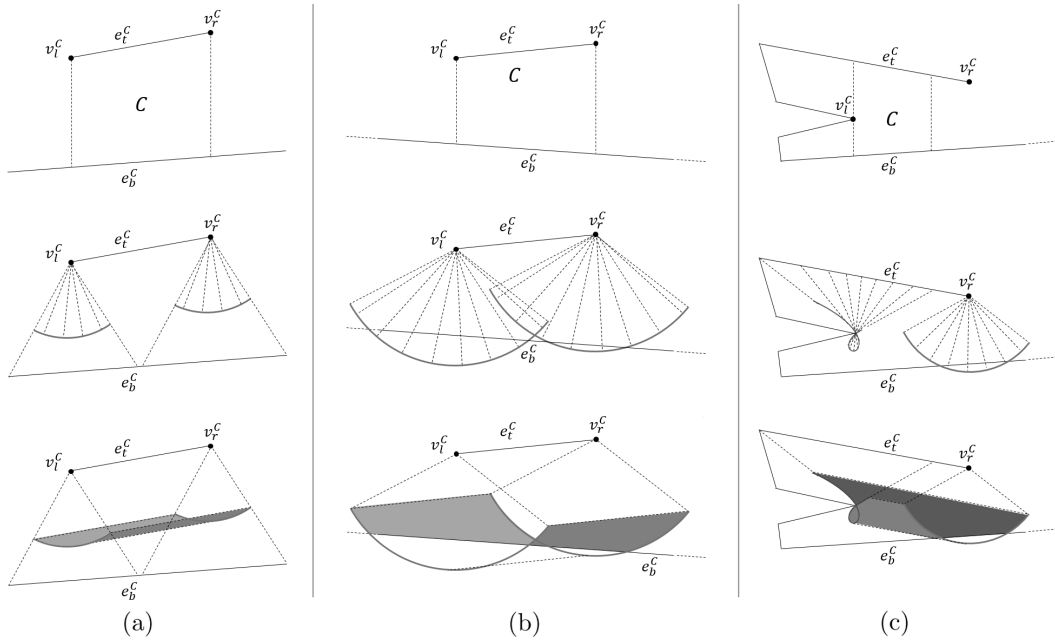
Figure 3.3: Examples of the planar projection of possible configurations of a single cell with a fixed reading d: (a) Both limiting vertices lie on the top edge, the resulting curves are circular arcs and do not intersect the bottom edge. (b) The limiting vertices and arcs are similar to (a), but the result intersects the bottom edge. (c) One limiting vertex lies on the top edge and one does not, resulting in a circular arc and a conchoid.

3.3   Suboptimality of the RTD-Based Solution

The RTD-based data structure is output sensitive with respect to the RTD decomposition, and in this section we explain why the data structure is not output sensitive with respect to an optimal algorithm. In fact the gap from optimality is large: $\Theta(n)$ as shown in the following example.

Consider Figure 3.4, where the workspace is a regular (convex) polygon with n edges, and the sensor reading is a value d, which is small relative to the diameter of the workspace. The combinatorial complexity of the union of all the possible configurations yielding a reading d is $\Theta(n)$, whereas the sorted-list structure will report an output of size $\Theta(n^2)$. The reason is that all the answers with a fixed ceiling edge e are fragmented by the RTD decomposition into $\Theta(n)$ cells each, while d is too small to be affected by the floors of most of these cells, and an optimal output-sensitive answer to the query would have reported the answer per fixed ceiling edge in $O(1)$ fragments, each of constant size.
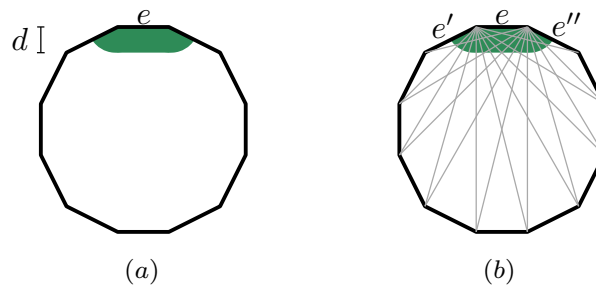
Figure 3.4: (a) An example of a regular (convex) polygon workspace with n vertices, and a given distance measurement d. The possible configurations for a sensor's measuring ray to hit one of the edges, denoted e, is shown in green. The complexity of that area is $O(1)$ and therefore the result for the whole workspace can be described using $O(n)$ space. (b) The RTD cells with top edge e are shown, there are $\Theta(n)$ of them. Each one of these cells will have a non-empty result. Therefore the complexity of the result for the whole workspace will be $\Theta(n^2)$, which is greater than the optimal by a factor n.

Chapter 4

Output-Sensitive Data Structure for Single Measurement

We described a data structure at Chapter 3, which is output sensitive *with respect to the RTD*, but as mentioned in Section 3.3, it is far from answering queries optimally. In this chapter we present a more involved data structure, which provides output-sensitive answers to single-measurement queries. This data structure is one of the major contributions of this thesis, along with the output sensitive data structure for antipodal measurements, described below in Section 5.3.

An edge e of the workspace and a fixed distance d induce a ruled surface $\Sigma(e, d)$, which is the locus of all points $(p, \theta) \in \mathbb{R}^2 \times \mathbb{S}^1$ such that there is a segment $\overline{pq}$ in $\mathbb{R}^2$ of length d, with $q \in e$, such that the vector $\overrightarrow{pq}$ makes an angle θ with the positive x-axis, and such that $\overline{pq}$ is contained in the half-plane defined by the line supporting e and locally near e facing the interior of the workspace. We consider e to be one-sided—we are only interested in the side of e facing the interior of the workspace. The union of points on $\Sigma(e, d)$ for which $\overline{pq} \in W$, defines a collection of surface patches, which we denote by $\sigma(e, d)$. For a single measurement d, and for every edge e on the boundary of W, we need to report $\sigma(e, d)$ unless $\sigma(e, d) = \varnothing$. The combinatorial complexity of these surface patches is the total number of vertices, edges (maximal arcs of low degree algebraic curves) and faces that define the patches. This complexity is a lower bound on the query time of any algorithm for computing the required preimage.

We will build the data structure by varying d from 0 to D, where D is the diameter of the workspace, and updating the result $\sigma(e, d)$ only when it changes combinatorially, for any edge e. The vertices and edges of the result of $\sigma(e, d)$ will be stored in an interval tree [8, Chapter 10] $\Delta(e)$, where the interval is the set of d values for which the specific result edge prevails, so it is possible to retrieve the relevant features for any query d value.

4.1 Result Vertices, Edges and Their Changes

Fix an edge e. We now embed critical surfaces in $\mathbb{R}^2 \times \mathbb{S}^1$ such that for a given query value d, they will induce a subdivision on $\Sigma(e, d)$, some of whose faces and *all* its vertices and edges, constitute the desired $\sigma(e, d)$. For each vertex in the locally-free half-plane of e let $\chi_e(v)$ be the surface which is the union of the following rays: for every θ, the ray emanating from v in direction $\theta + \pi$. For each edge e' let $\psi(e')$ be the surface which is simply the edge e' for every θ, namely it is a vertical rectangle in $\mathbb{R}^2 \times \mathbb{S}^1$.

We call the vertices and edges of $\sigma(e, d)$ the *result vertices* and *result edges* (for edge e and measurement d) to avoid ambiguity with the workspace features. The result vertices are created by intersecting $\Sigma(e, d)$ with two critical surfaces. By the general position assumption, at most two surfaces $\chi_e(v_i)$ intersect in a single point—in fact in a single ray. Similarly, at most two surfaces $\psi(e_i)$ intersect at a point (in a vertical line segment). Two surfaces $\psi(e_1), \psi(e_2)$ can intersect $\Sigma(e, d)$ at the same point, but in such case there will be another surface $\chi_e(v)$ involved where v is the workspace vertex, which is the common endpoint of e_1 and e_2. Therefore, the result vertices are obtained in two types of intersections:

- Type (i): Two $\chi_e(\cdot)$ surfaces intersect $\Sigma(e, d)$.

- Type (ii): A single $\chi_e(\cdot)$ surface intersects $\Sigma(e, d)$, along with some other $\psi(\cdot)$ surface.
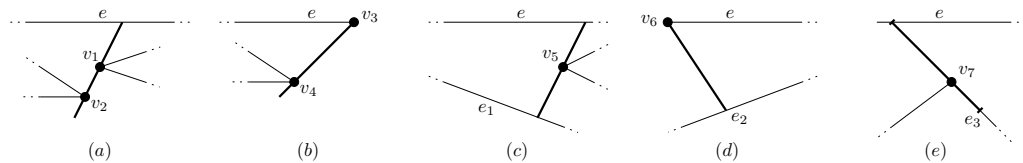

Figure 4.1: Examples of result vertices of type (i) in (a),(b) and of type (ii) in (c),(d),(e). The bold segment has length d, and represents a single point in the configuration space, which is a vertex in the result $\sigma(e, d)$. A result vertex created by surfaces of two vertices of the workspace (a), where one of them can be an endpoint of e (b).

The overall scheme for preprocessing a single edge e is as follows. We will sweep the surface $\Sigma(e, d)$ over the space $\mathbb{R}^2 \times \mathbb{S}^1$, from $d = 0$ to $d = D$, where D is the diameter of the workspace W. (Notice that the sweeping surface itself changes throughout the process; however, it changes continuously.) At all times during the spatial sweep we will record the edges that appear on the boundary of $\sigma(e, d)$ in an interval tree $\Delta(e)$. The exact geometric curve of each edge is not explicitly stored as it changes as d varies, rather the workspace features that define the curve are stored, which enables us to compute it exactly for a specific d in $O(1)$ time, together with the attribute on which side of an edge does the result lie. Given a query measurement d, using simple auxiliary data structures to be described later, we will efficiently retrieve the k_e edges of $\sigma(e, d)$, and in $O(k_e \log k_e)$ time will construct a DCEL describing the result $\sigma(e, d)$. The events of the sweep are those where the surface patches $\sigma(e, d)$ change combinatorially. These in turn happen at all the minimum and maximum values of d in which each result vertex exists. There are five types of such events:

(I) The minimum value of d at which a result vertex of type (i) involving two vertices v_1, v_2 exists is the length of the line segment from v_2 to e which goes through v_1, assuming v_2 is further away from e than v_1. See Figure 4.2(a).

(II) The maximum value of d in which a result vertex of type (i) involving two vertices v_1, v_2 exists is the length of the maximal line segment passing through v_1, v_2, hitting e at one end and another edge of the workspace at its other end. See Figure 4.2(b).

(III) The minimum value of d in which a result vertex of type (ii) involving a vertex v_3 and an edge e_1 exists. If v_3 is the closer endpoint of e_1 to e, d is the length of the segment starting at v_3 extending e_1 (along the line supporting e_1) and hitting e (see Figure 4.1(e)). If v_3 is a reflex vertex of the workspace that is not an endpoint of e_1, then d is the length of the minimum line segment s passing through v_3, hitting e and e_1 at its endpoints—this is the unique such segment that makes an equal angle with both edges, i.e., $\measuredangle se = \measuredangle se_1$ (see Figure 4.2(c)).

(IV) The two maximal values of d in which a result vertex of type (ii) involving a vertex v_3 and an edge e_1 exists are the maximal lengths of a line segment passing through v_3, hitting e and e_1 at its endpoints—there are two such maximal lengths. See Figure 4.2(d,e).

(V) The distance of a vertex v to e, when the shortest line segment from v to e is contained in the workspace. See Figure 4.2(f).
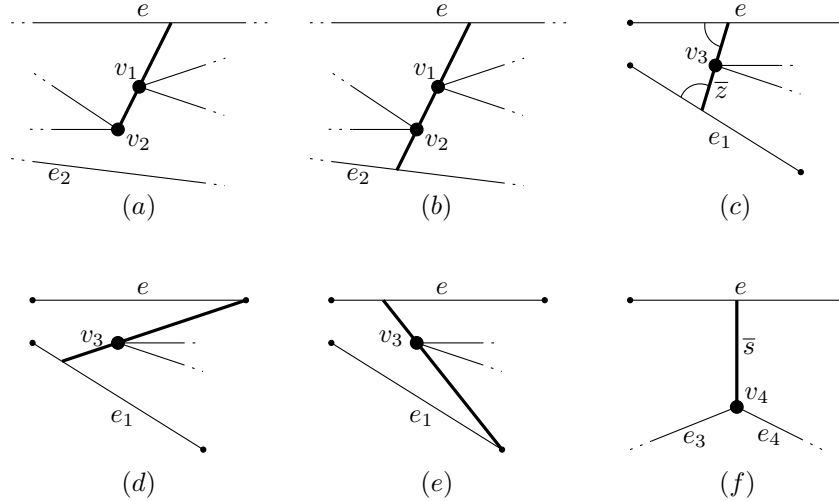
Figure 4.2: Examples of configurations and values of d in which the result changes combinatorially. (a)(b) The unique minimum and maximum values of d in which $\Sigma(e,d), \chi_e(v_1), \chi_e(v_2)$ create a result vertex. (c)(d)(e) The unique minimum and two maximal values of d in which $\Sigma(e,d), \chi_e(v_3), \phi(e_1)$ create a result vertex. The minimum is achieved when the segment of length d makes two equal angles with e, e_1. (f) The minimum value of d in which $\Sigma(e,d), \chi_e(v_4), \phi(e_3)$ and $\Sigma(e,d), \chi_e(v_4), \phi(e_4)$ induce result vertices.

We wish to find all the values of d for which the result $\sigma(e,d), \forall e$ is changing combinatorially when d is varying from 0 to D, the diameter of W. To that end we carry out a global

19

preprocessing stage, where we collect *combinatorial change tuples* of the form (e, θ, d, U), where d is a threshold in which the result of edge e is changing combinatorially, and U is the set of workspace vertices involved in the change. Collecting all these tuples is done similarly to computing the RTD, by carrying out radial sweeps through all the vertices of the workspace, considering only pairs of visible vertices by computing the visibility graph in $O(E + n \log n)$ time (where E is the number of edges in the visibility graph of W), and requires $O((E+n) \log n)$ time and $O(E+n)$ space. The details of these sweeps are technical, and we defer their full description to Section 4.4, where it is also shown that at most $O(E)$ such tuples are exist and indeed collected at the preprocessing stage.

4.2   Combinatorial Result Change Handling

All the tuples with the same first parameter e, sorted in increasing d value, are the events of the spatial sweep with the surface $\Sigma(e, d)$. We compute the result $\sigma(e, d)$ for infinitesimal values of d, by considering only e and its two adjacent edges. From this point on, result edges will be inserted or removed only when handling combinatorial change tuples. We handle each combinatorial change (e, θ, d, U) according to the event type. The details of these handling is technical, and we focus on the handling of type I changes in the this section. The full description of the handling of changes is given in Appendix B.

Combinatorial change of type I : d is the minimum value in which a result vertex of type (i) involving two vertices v_1, v_2 exists (namely an intersection of two $\chi_e(\cdot)$ surfaces intersects $\Sigma(e, d)$), where v_1 is the vertex closer to e. We say that a vertex is a *reflex vertex relative to a line (resp. segment)* if both of its incident edges are in the same half-plane created by the line (resp. the line supporting the segment). We will assume that v_2 is a reflex vertex relative to $\overline{v_1 v_2}$; otherwise, d is also the maximum value in which such a result vertex exists, and we will handle it as a change of type II (see Figure 4.4(a) for such a scenario). v_1 is clearly a reflex vertex relative to $\overline{v_1 v_2}$. Let e_1 be the edge of v_2 visible from v_1. We say that a result edge η' is defined by e and a vertex v when η' is the union of configurations where a rod of length d is sliding on e and v. We say that a result edge η'' is defined by e and another edge e_1 when η'' is the union of configurations where a rod of length d is sliding with one endpoint on e and another endpoint on e_1. The limits α, β of the angle range $[\alpha, \beta]$ throughout which a result edge prevails, can be specified by a fixed angle each, or by the labels of the workspace features that induced these angles for a specific d value. The handling operations depend on two predicates: whether the edges of both v_1, v_2 are in the same halfplane relative to $\overline{v_1 v_2}$ or not, and whether a result edge defined by e, e_1 exists or not in $\sigma(e, d')$ for d' is infinitesimally smaller than d. The sub-cases are illustrated in Figure 4.3. We present here the operations performed in one sub-case: The edges of both v_1, v_2 are in the left halfplane relative to $\overline{v_1 v_2}$, and a result edge η_1 defined by e, e_1 exists in the angle range $[\alpha, \gamma]$ for some α, where γ depends on d and it is defined as the angle that the line segment passing through v_1 with varying length d with its endpoints on e, e_1 makes with the x-axis.

There must be another result edge η_2 defined by e, v_1 in the angle range $[\gamma, \beta]$ for some β. We perform the following operations:

- Find and remove η_1 and η_2.
- Add a result edge defined by e, v_2 in the angle range $[\delta, \theta]$ where $\delta \leqslant \theta$ is the angle of the line segment with varying length d with its endpoints on v_2 and e (there are two such segments and we choose one of them by the constraint $\delta \leqslant \theta$). Recall that θ is a parameter of the combinatorial change tuple.
- Add a result edge defined by e, v_1 in the angle range $[\theta, \beta]$.
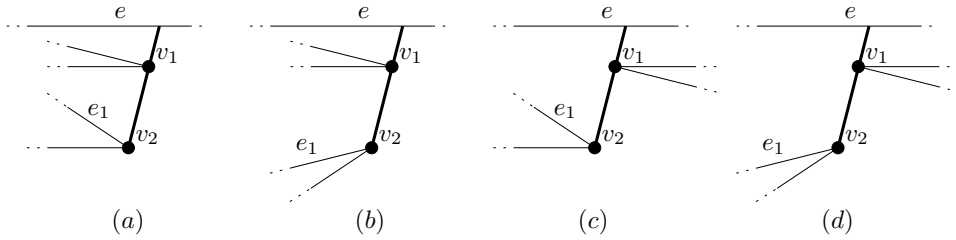

Figure 4.3: Examples of the sub-cases arising in the handling of combinatorial changes of type I. The bold line segments are each of length d. There is a result edge defined by e, e_1 for d' infinitesimally smaller than d in (a),(c) and there is no such result edge in (b),(d). Edges of both v_1, v_2 are on the same halfplane relative to $\overline{v_1 v_2}$ in (a),(b) and in opposing sides in (c,d).

It is important to note that we do not maintain face objects of the subdivision $\sigma(e, d)$,and we only maintain the edges of this subdivision. This allows us to handle combinatorial changes of type V by creating a new hole inside an existing face without locating the enclosing face. For handling most combinatorial change types, it is required to find an existing edge in the list of current edges on $\sigma(e, d)$, based on the tuple information. To that end, we maintain auxiliary balanced search trees for each pair of edge and vertex $T(e_i, v_j)$ for all e_i and v_j, and $T(e_k, e_\ell)$ for each pair of edges e_k, e_ℓ. The search trees $T(e_i, v_j)$, $T(e_k, e_\ell)$ will contain all the edges currently in $\sigma(e, d)$ that are defined by (e_i, v_j) and (e_k, e_ℓ), respectively. The key for the auxiliary trees is the smallest angle in the angle interval in which the edge exists. The angles are not explicitly stored in the tree (since they change as d varies), rather only the order of the edges is maintained. This is possible because multiple edges defined by the same features exist in disjoint ranges of angles, and their order does not change. When handling a combinatorial change, the result edges can be found using simple $O(\log n)$-time search with the angle θ in the relevant trees, and newly created edges are inserted into these auxiliary data structures, in $O(\log n)$ time total.

4.3   Storing the Result Edges for Efficient Queries

For each edge e, the interval tree $\Delta(e)$ is constructed independently using the associated combinatorial changes. Let ν_e be the number of combinatorial changes involving the edge e. Construction of a single data structure $\Delta(e)$ requires $O(\nu_e \log n)$. The overall number of

events over all edges is $O(E)$ and therefore it takes total $O(E \log n)$ time to construct the data structures for all the edges altogether. Finally, we also compute for every edge e the maximum d for which $\sigma(e, d) \neq \varnothing$; we denote this maximum by $m(e)$. We will then keep these interval trees maxima sorted in decreasing order in a list L. The preprocessing requires $O((E + n) \log n)$ time and $O(E)$ space in total.

During a query for a single measurement d, we iterate over the data structures in L by their sorted order as long as $m(e) \geqslant d$, ensuring each data structure we examine will have a non-empty result, excluding possibly the last one. (Notice that if a structure has non-empty result for d' then it also has a non-empty result for any $d < d'$.) For each data structure $\Delta(e_i)$ that has a non-empty result, we query the interval tree for the result edges, in time $O(\log n + k_i)$, where k_i is the number of result edges found. For each of result edge, we compute its exact arc, which depends on the query specific d value. Then using a standard algorithm,[1] we construct the complete desired DCEL in $O(k \log k)$ time, where $k = \Sigma_{i=1}^{n} k_i$ is the number of result edges in total. The result faces can be identified since, as noted above, each edge stores which of its sides (halfedges in the DCEL) is contained in the result. The total running time of the query is $O(1 + k \log n)$.

Theorem 4.3.1. *Given a polygonal workspace with a total of n vertices, a data structure can be constructed in time $O((E + n) \log n)$ and $O(E + n)$ space, where E is the number of edges in the visibility graph of W, such that, given a query real parameter $d > 0$, we can report all possible configurations yielding depth measurement d in time $O(1 + k \log n)$, where k is the number of vertices and maximal arcs of low algebraic degree appearing on the boundary of the surface patches that constitute the result.*

4.4   Computing the Combinatorial Change Tuples

We wish to find all the values of d for which the result $\sigma(e, d), \forall e$ is changing combinatorially when d is varying from 0 to D, the diameter of W. To that end we carry out a global preprocessing stage, where we collect *combinatorial change tuples* of the form (e, θ, d, U), where d is a threshold in which the result of edge e is changing combinatorially, and U is the set of workspace vertices involved in the change.

Collecting all these tuples is done similarly to computing the RTD. First, the visibility graph of W is computed in $O(E + n \log n)$ time [18], where E is the number of edges in the visibility graph of W. Let $\mathcal{E}$ denote the set of edges in the visibility graph of W. Then, we sort the set of ordered pairs $\bigcup_{\{u,v\} \in \mathcal{E}} \{(u, v), (v, u)\}$ by the slope of the line oriented at v_1 directed at v_2 of each ordered pair (v_1, v_2), and perform from each vertex v of the workspace a rotational plane sweep, in two antipodal directions simultaneously. The other visible vertices of the workspace are the events of the sweep, and the first edge intersecting each imaginary sweep ray is the state of the sweep. (There is no need to store all edges that intersect the ray, as we already know the visibility graph). Consider an event at a vertex $u \neq v$. The

---

[1]See [8, Section 2.3] for details; it is described there for completing the overlay of two subdivisions, but applies equally well here after a preprocessing step where we form boundary cycles of the DCEL.

event is handled differently depending on whether u and v are *reflex* vertices relative to the line $\overline{vu}$. Recall that we say that a vertex is a *reflex vertex relative to a line* if both of its edges are in the same halfplane bounded by the line.

We will cover here in details the case where u is a reflex vertex relative to $\overline{vu}$ and v is not; the other cases are handled in an analogous fashion: Let e_1, e_2 be the edges to the right and left of v, respectively, relative to the direction $\overrightarrow{uv}$. Let e_3 be the edge of u that is visible from v and let e_4 be the edge that the ray originating at v in the direction $\overrightarrow{vu}$ hits after passing through u (see Figure 4.4(b)). Let θ be the angle of the direction $\overrightarrow{vu}$, relative to the x-axis, let $dist(v, u)$ be the distance between v and u, and let $dist(v, u, e_4)$ be the length of the segment originating at v, passing through u and hitting e_4. Add the following combinatorial change tuples:

$$(e_1, \theta + \pi, dist(v, u), \{v, u\}), \quad (e_1, \theta + \pi, dist(u, v, e_4), \{v, u\}),$$
$$(e_2, \theta + \pi, dist(v, u), \{v, u\}), \quad (e_2, \theta + \pi, dist(u, v, e_4), \{v, u\}),$$
$$(e_3, \theta, dist(v, u), \{v, u\}), \quad (e_4, \theta, dist(u, v, e_4), \{v, u\}).$$

The above six combinatorial change tuples are the changes of types that include two vertices, namely changes of types I, II, and IV. To find the changes including only a single vertex, during the rotational sweep around the vertex v, we consider the vertices u_1, u_2 of two consecutive events. Let θ_1, θ_2 be the angles the directed lines $\overrightarrow{vu_1}, \overrightarrow{vu_2}$ make with the positive x-axis, respectively. Denote by e_1, e_2 the edges that are hit by the two antipodal rays of the sweep for any angle $\theta \in (\theta_1, \theta_2)$ and $\theta + \pi$ respectively. Denote by $\overline{z}$ the line segment that passes through v and makes equal angles when hitting e_1, e_2. The segment $\overline{z}$ is directed from its endpoint on e_1 to its endpoint on e_2. If the angle $\measuredangle \overline{z}$ that the segment $\overline{z}$ makes with the positive x-axis, is in the range (θ_1, θ_2), add the combinatorial tuple change of type III $(e_1, \measuredangle \overline{z}, ||\overline{z}||, \{v\})$, where $||\overline{z}||$ is the length of $\overline{z}$ (see Figure 4.2(c) for illustration). Denote by $\overline{s}$ the shortest line segment between v and e_1. If the angle $\measuredangle \overline{s}$ that $\overline{s}$ makes with the x-axis is in the range (θ_1, θ_2), add the combinatorial tuple change of type V $(e_1, \measuredangle \overline{s}, ||\overline{s}||, \{v\})$ (see Figure 4.2(f) for illustration).
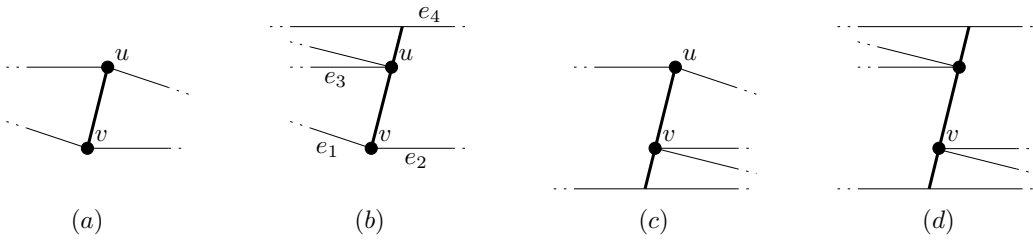

Figure 4.4: Examples of scenarios arising when handling an event of vertex u during a rotational plane sweep around a vertex v. (a) Both u and v are not reflex vertices relative to $\overline{uv}$. (b)(c) Only one of u, v is a reflex vertex relative to $\overline{uv}$. (d) Both u and v are reflex vertices relative to $\overline{uv}$.

We perform such a rotational sweep around each vertex of the workspace and add all the combinatorial change tuples. Each time an event is handled, a constant number of combinatorial change tuples are added, therefore the total number of such combinatorial change tuples is $O(E)$. The time complexity of this preprocessing stage is $O((E + n) \log n)$.

Chapter 5

Localization by Antipodal Measurements

An important variant of the problem is a sensor that takes two different distance measurements from the same location, in opposite directions. We obtain one measurement d_1 in (unknown) direction θ_1 and a second measurement d_2 in direction $\theta_2 = \theta_1 + \pi$. We seek to efficiently compute the intersection of the C-space preimages that correspond to each measurement. A nice, mitigating property of this setting is that the two distance measurements are necessarily taken to two *distinct* edges of the workspace.

Two data structures are presented in this chapter: (i) a simpler, RTD-based data structure, which is output sensitive with respect to the RTD, and (ii) an output sensitive data structure, which uses the RTD cells as intermediate building blocks, but then modifies the decomposition cells by unions and splits of the cells into a more economical collection. The second structure (Section 5.3) is the other major contribution of this thesis, along with the output sensitive data structure for a single measurement, described before in Chapter 4.

5.1 RTD-Based Data Structure for Antipodal Measurements

In this section, a solution based on the decomposition described in Chapter 2 is used, where the orientation θ is the orientation of the first measurement. The data structure achieves a query time of $O(\log n + k)$, where k is the number of RTD cells that contains a result configuration.

For a fixed orientation θ, the cross-section of an RTD cell C contains one potential sensor pose (assuming the top and bottom edges of the cell are not parallel), and we can calculate it by finding the segment of length $d_1 + d_2$ with its endpoints on the cell's top and bottom edges. Which in turn we do by solving $O^C(\theta, x) = d_1 + d_2$, and extracting the point along the segment obtaining this opening that is at distance d_1 from the top edge (and hence at distance d_2 from the bottom edge). For each such cell C we compute the locus $\Gamma(C)$

of potential sensor poses, which is a curve parameterized by the orientation of the first measurement.

In order to store the cells of the decomposition so as to be able to efficiently retrieve the cells C with non-empty loci $\Gamma(C)$, we now need to also compute the minimum opening of a cell O_{min}^C, which is defined analogously to O_{max}^C (cf. Chapter 3). Given a pair of antipodal measurements (d_1, d_2), a cell C has a non-empty set of loci if and only if $O_{min}^C \leqslant d_1 + d_2 \leqslant O_{max}^C$. Each cell is associated with an interval $[O_{min}^C, O_{max}^C]$, and we construct an interval tree over these $O(E)$ intervals. Given the measurements (d_1, d_2), we extract all the intervals that contain the value $d_1 + d_2$, and their corresponding cells contribute potential poses to the final answer.

The loci $\Gamma(C)$ constitute an arc of an ellipse with its center at the (possibly imaginary) intersection of the lines supporting the top and bottom edges of the cell; see Appendix A.2 for details. After prepossessing in $O((E + n) \log n)$ time and using $O(E + n)$ space, a query can be answered in $O(\log n + k)$ time, where k is the number of cells C with non-empty $\Gamma(C)$.

Theorem 5.1.1. *Given a polygonal workspace W with a total of n vertices, an interval tree containing the RTD cells of the configuration space can be constructed in time $O((E+n) \log n)$ and space $O(E)$, where E is the number of edges in the visibility graph of W, which is bounded by $O(n^2)$. Then, given two real numbers $d_1, d_2 > 0$, we can report all the RTD cells that contain a possible configuration yielding antipodal depth measurement d_1, d_2 in time $O(\log n + k)$ where k is the number of the relevant cells.*

5.2 Suboptimality of the RTD-Based Solution

The data structure presented at Section 5.1 is output sensitive with respect to the RTD decomposition, and in this section we demonstrate why it is not output sensitive with respect to an optimal algorithm. In fact the gap from optimality is large: $\Theta(n)$ as shown in the following example.

Consider Figure 5.1, where the workspace is a triangle-like workspace with additional $Theta(n)$ reflex vertices on the left edge. The angle interval in which a sensor can measure the edges e_1 and e_2 with two antipodal measurement is fragmented by the RTD decomposition into $\Theta(n)$ cells of the form $(e_1, e_2, v_i, u, \cdot)$ where v_i is a different reflex vertex as a limiting left vertex of the cell. Although this fragmentation may be needed for some values of d_1, d_2, for smaller such values (as shown in the figure) the change of the left limiting vertex does not affect the result. There will be $\Theta(n)$ RTD cells with e_1, e_2 as top and bottom edges that will contribute to the result, while the output can be described using only $O(1)$ space.
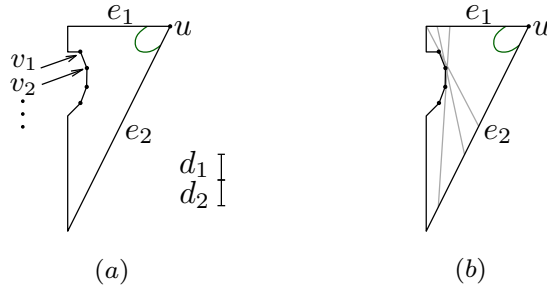
Figure 5.1: A triangle-like workspace, with additional $\Theta(n)$ reflex vertices on the left edge. Given a query of small $d_1 = d_2$, the possible positions a sensor can measure e_1, e_2 at distances d_1, d_2 respectively are drawn as a green arc at the top-right corner of the workspace. RTD will contain $\Theta(n)$ cells with e_1, e_2 as top and bottom edges, and u as the right limiting vertex, each with a different left limiting vertex v_i, shown in (b). Each of these cells will have a non-empty result, while the output of these two edges has $O(1)$ descriptive complexity.

5.3 Output-Sensitive Data Structure for Antipodal Measurements

The result of a query of two measurements (d_1, d_2) can be naturally and economically described by a set of angle intervals $I_{e_1, e_2} = \{[\alpha_1, \beta_1], [\alpha_2, \beta_2], \ldots\}$ for any relevant pair of workspace edges e_1, e_2, where for each interval the sensor reading will hit e_1 at any angle $\theta \in [\alpha_i, \beta_i]$ and e_2 at angle $\theta + \pi$, and there is only one such configuration (assuming, as we do, that there are no parallel edges in the workspace). When considering this representation, the minimum complexity required is achieved when the intervals are maximal and disjoint.

For a fixed maximal interval $[\alpha_i, \beta_i] \in I_{e_1, e_2}$, we consider the line segment of length $d = d_1 + d_2$ at angle α_i with its endpoints lying on e_1 and e_2. This line segment must also intersect a vertex of the workspace, either an endpoint vertex of e_1 or e_2, or some other reflex vertex of the workspace, as otherwise the angle interval would not be maximal. The other direction also holds: The existence of a line segment of length d passing through a vertex with its endpoints lying on two different edges e_3, e_4 at angle θ, must imply that the angle θ is included in one of the result angle intervals of e_3, e_4, and it must be an extreme angle in that interval. See Figure 5.2 for an illustration. Therefore the minimum complexity required to describe a query result for antipodal measurements (d_1, d_2) is the number of line segments of length $d = d_1 + d_2$ that can be placed in the interior of the workspace such that both their endpoints lie on edges of the workspace and additionally they intersect a vertex of the workspace.

To construct a data structure that will provide such answers, we will use the RTD of the C-space in a preprocessing stage, after which we will merge results from different cells of the RTD, wherever possible. For each RTD cell C, we define the minimum left opening O_{min}^{Cl} as the minimum opening obtained over all the angles that C exists in with the x-coordinate of
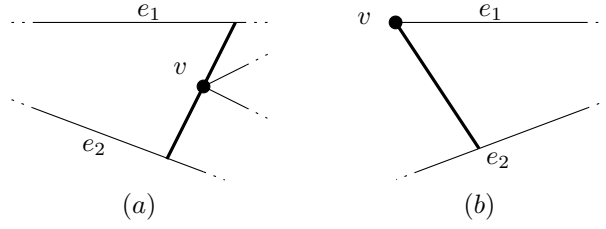
Figure 5.2: Examples of configurations which are vertices in the minimum-complexity result for two antipodal measurements, where the bold line segment has length $d_1 + d_2$.

the top left vertex of C. Similarly, we define minimum/maximum left/right openings:

$$O_{min}^{Cl} = \min_{\theta \in \Theta^C} O(\theta, x_{lt}^C(\theta)) \;, \quad O_{max}^{Cl} = \max_{\theta \in \Theta^C} O(\theta, x_{lt}^C(\theta)) \;,$$

$$O_{min}^{Cr} = \min_{\theta \in \Theta^C} O(\theta, x_{rt}^C(\theta)) \;, \quad O_{max}^{Cr} = \max_{\theta \in \Theta^C} O(\theta, x_{rt}^C(\theta)) \;.$$

Based on the relation between d and the minimum/maximum left/right openings, we classify each RTD cell C as one of three classes: *Empty* class Em, *Full* class F or *Partial* class P:

$$Em = \{C \mid d < \min\{O_{min}^{Cl}, O_{min}^{Cr}\} \; \vee \; d > \max\{O_{max}^{Cl}, O_{max}^{Cr}\}\} \;,$$

$$F = \{C \mid O_{max}^{Cr} < d < O_{min}^{Cl} \; \vee \; O_{max}^{Cl} < d < O_{min}^{Cr}\} \;,$$

$$P = \{C \mid O_{min}^{Cl} < d < O_{max}^{Cl} \; \vee \; O_{min}^{Cr} < d < O_{max}^{Cr}\} \;.$$

The members of the class Em are *empty* in the sense that each RTD cell in this class does not contain any valid result, as the opening is either greater or smaller than d for any angle in the cell's angle interval. The members of the class F are *full* in the sense that each RTD cell C in this class contributes the full angle interval Θ^C to the result, since the right (resp. left) opening is always greater than d and the left (resp. right) opening is always smaller than d, and due to the fact the opening function is linear for a fixed angle (see Section 2.2) this implies that there is some point between the left and right vertices where the opening is equal to d, for any angle in Θ^C. The members of the class P are *partial* in the sense that each RTD cell C in that class contributes a part of the angle interval Θ^C to the result, since there is some angle θ at which the right or left opening exactly equals d, which means that at angle $\theta + \epsilon$ (or $\theta - \epsilon$) for some small ϵ the opening for any point in the cell is not equal to d.

Another aspect to consider is that each cell C in the partial class contains a configuration which will be a vertex in the result, as a segment of length d can be placed such that its endpoints lie on e_t^C and e_b^C exactly on the vertices v_{lt}^C, v_{rt}^C, and one of these vertices must be a workspace vertex. For each pair of edges, the angle intervals contributed by the cells of full and partial classes are pairwise interior disjoint, but may overlap at their endpoints. By merging the angles intervals of each pair of edges to maximal intervals, the minimum-complexity output is achieved.

Let $\mathcal{D}$ denote the set of all minimum/maximum left/right opening of all the RTD cells,

along with ∞ and 0, sorted in increasing order:

$$\mathcal{D} = [d_0, d_1, \ldots \mid d_i < d_{i+1}, d_i \in \{0, \infty\} \cup \bigcup_C \{O_{min}^{Cl}, O_{max}^{Cl}, O_{min}^{Cr}, O_{max}^{Cr}\}] \ .$$

For two values d', d'' the RTD cells classification, and therefore also the result angle intervals, are combinatorially equivalent if there exists i such that $d', d'' \in [d_i, d_{i+1})$. We can perform the classification of the RTD cells and the union of the angle intervals for each range $[d_i, d_{i+1})$ during preprocessing, which will result in angle intervals with symbolic endpoint values, which in turn can be made concrete for any fixed value of d. During query with d_1, d_2, we select the result corresponding to the range $[d_i, d_{i+1})$ such that $d_1 + d_2 \in [d_i, d_{i+1})$, which contains maximal angle intervals and therefore optimal in size, and calculate their exact values for the specific value $d = d_1 + d_2$.

Producing and storing the full result for each range $[d_i, d_{i+1})$ is expensive. For efficiency we will again use an interval tree to store the result maximal angle intervals. The values stored in the interval tree are the result angle intervals, and the keys of these values are the ranges of d for which the values (the angle intervals) are present in the result. Although the values are intervals themselves, they have nothing to do with the interval keys of the tree, which are, again, ranges of d.

Such an interval tree can be constructed in the following way. We vary d from ∞ to 0 by iterating over the ranges $[d_i, d_{i+1}]$. The result of the current value of d, which is a set of maximal angle intervals, is maintained at all times as a balanced search tree R. Each result angle interval is stored in R using a key (e_1, e_2, θ_1), where e_1, e_2 are the two edges the sensor could have measured, and θ_1 is the minimal angle in the result interval. This representation of the result set allows us to efficiently find the result angle intervals that are affected combinatorially when we change d. We iterate over the ranges of d in order, computing the result intervals of $[d_i, d_{i+1}]$ based on the result intervals of $[d_{i+1}, d_{i+2}]$. We start with the range $[d_{|D|-1}, d_{|D|}] = [d_{|D|-1}, \infty]$, in which all RTD cells are classified as *empty* and the result contains no intervals, $R = \varnothing$. Assuming we already classified the RTD cells and calculated the result intervals for the range $[d_{i+1}, d_{i+2}]$, we assume that a single cell C changes its class when considering the next range $[d_i, d_{i+1}]$. If multiple cells change their classes, we repeat the following process multiple times.

- If C was full or partial relative to the previous range $[d_{i+1}, d_{i+2}]$, we remove the angle interval it contributes from R. We may have to split a larger interval that was created from multiple intervals to remove it.

- If C is full or partial relative to the current range $[d_{i+1}, d_i]$, we add the angle interval it contributes to R. We may have to merge it with at most two additional intervals that intersect it at its endpoints.

In both cases, a constant number of insertions, deletions, splits and merges are performed. For each interval, we store the values of d in which it was added or removed from R, or 0 if it was not removed after iterating over $[d_0, d_1] = [0, d_1]$. We insert all such angle intervals into an interval tree keyed by the range of values of d for which each angle interval was in R.

In total, the number of operations performed to build the data structure is linear in the number of RTD cells, which is $O(E)$. Each operation on R requires $O(\log n)$ time, and constructing an interval tree with $O(E)$ elements requires $O(E \log n)$ time and $O(E)$ space. During query, the result intervals are retrieved from the interval tree in time $O(\log n + k)$, where k is the number of maximal angle intervals in the result, and the exact algebraic description can be computed for each one of them in $O(1)$ time.

Theorem 5.3.1. *Given a polygonal workspace W with a total of n vertices, a data structure can be constructed in time $O((E + n) \log n)$ and $O(E)$ space, where E is the number of edges in the visibility graph of W (which is bounded by $O(n^2)$), such that, given a query of two measurements (d_1, d_2), we can report all possible configurations yielding antipodal depth measurement d_1, d_2 in time $O(\log n + k)$ where k is the number of maximal elliptical arcs in the result.*

Chapter 6

Implementation and Experiments

We implemented the simpler data structures, which are built upon the rotational trapezoidal decomposition, both for single distance-measurement queries and for two-antipodal measurements queries (see Chapter 3 and Section 5.1). We describe our implementation details, challenges and report on some experiments that we conducted.

We implemented the structures in C++ using the CGAL library [31] and in particular the arrangements package [16]. The implementation is open source and available at `https://github.com/barakugav/FDML`. Given a workspace, a preprocessing is performed once, and multiple queries of both types can be answered with output-sensitive complexity with respect to the RTD. The implementation assumes general position, namely that the given workspace does not contain three vertices on the same line, three edges that intersect in a single point, etc. In addition to the C++ code, we also implemented Python bindings on top of CGAL's bindings [19], allowing for an easier use of our data structures. Results of queries for a single measurement and for two antipodal measurements are shown in Figure 6.1 and Figure 6.2. Notice that some workspace contains identical subregions, in which case a sensor located in one of these subregions can not distinguish between them, even if we allow for an unlimited number of measurements. See Figure 6.3 for an example of such an environment.
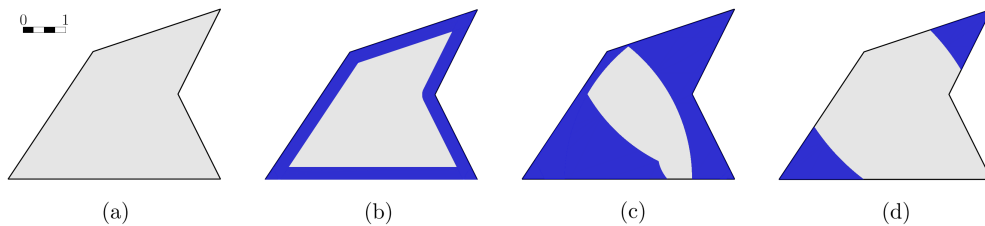

Figure 6.1: The possible positions of a sensor that performs a single measurement in a simple polygon workspace (a) are shown in blue in (b) for $d = 0.3$, in (c) for $d = 4$, and in (d) for $d = 5$.

Among the challenges we faced during the development of the software, we had to use exact number types to represent angles and points in the workspace. Although CGAL supports these number types using the GMP or MPFR libraries, their usage has a tremendous impact on performance. By using inexact number types such as 64-bit floating point, the
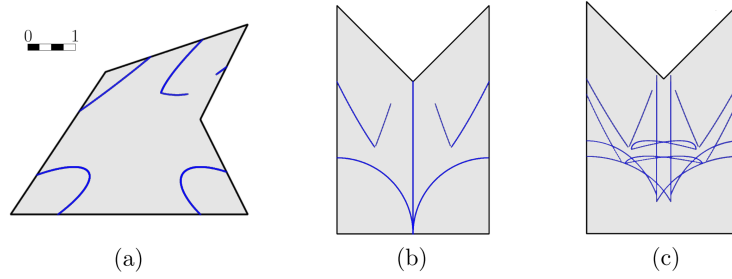
Figure 6.2: The possible positions of a sensor that performs two antipodal measurements in simple polygons are shown in blue in (a) for $d_1 = d_2 = 1$, (b) for $d_1 = d_2 = 1$ and (c) for $d_1 = 1, d_2 = 1.2$.

running time presented in Figure 6.5 can be improved by an order of magnitude. Another challenge was to represent the result curves (projected onto the plane), as these curves are of high degree (see Appendix A.1 and Appendix A.2). We approximate these curved using polylines, with parameterized precision.

We conducted a few additional performance experiments, measuring the preprocessing time and query time for a convex regular polygon with different number of vertices; see Figure 6.4. The experiments where run on a Windows 10 64-bit machine with 11th Gen Intel Core i7-1185G7 at 3.00GHz and 16GB memory. The results are shown at Figure 6.5.
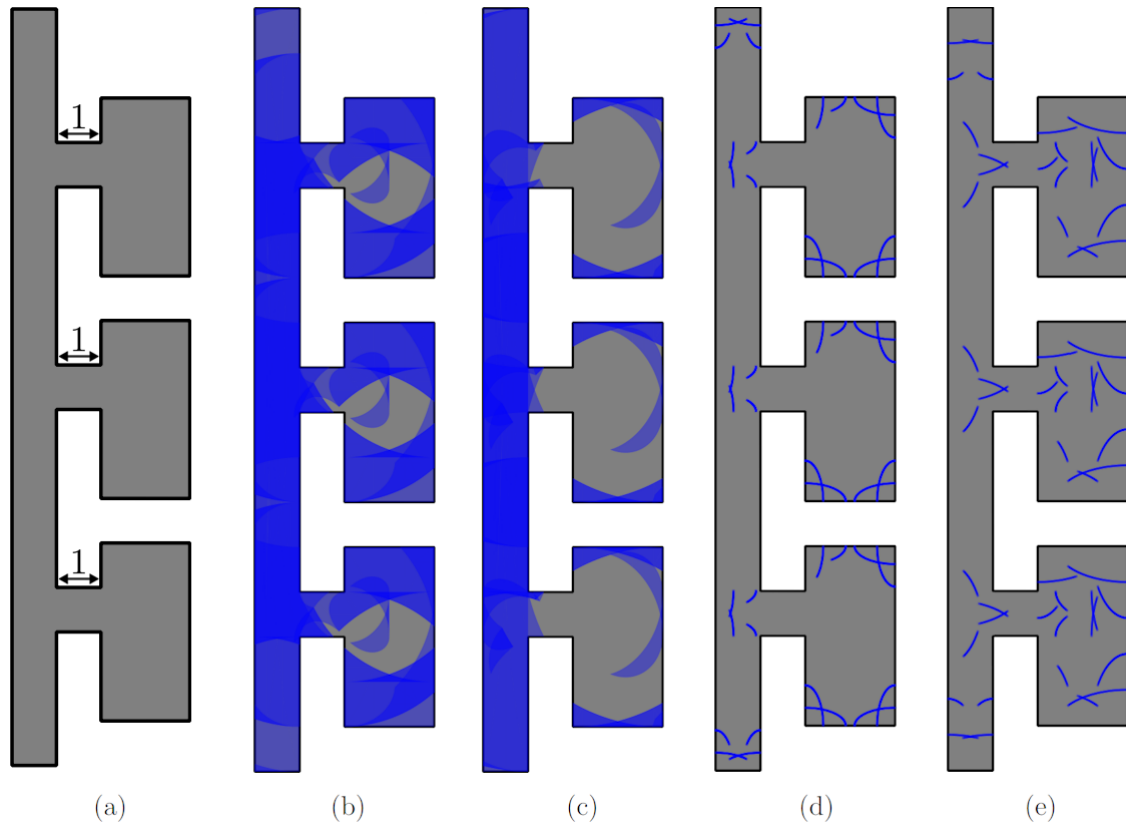
Figure 6.3: The workspace at (a) is a symmetric workspace; it comprises a corridor and three rooms, where the rooms are indistinguishable from one another for a sensor positioned in any one of them. In (b) and (c) the possible positions of the sensor are drawn, after it measured a single value of $d = 3$ and $d = 4$ respectively. In (d) and (e) the possible positions of the sensor are drawn, after it measured two antipodal measurements of $d_1 = 0.4, d_2 = 0.9$ and $d_1 = 0.8, d_2 = 1.6$ respectively. It is easy to see that if the robot is indeed located in one of the identical rooms it can not determine its position uniquely, no matter how many measurements it will perform. Notice that there is a range of intensity of blue color, where greater intensity corresponds to more RTD cells contributing a result to the region in question.
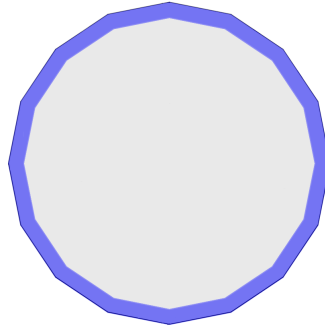
Figure 6.4:   An example of a regular convex polygon with 16 vertices and a radius of 1. The result of a query of a single distance measurement of 0.1 is a 'margin' area near the polygon boundary. The RTD decomposition decomposes the C-space for the regular polygon into $O(n^2)$ cells, each containing part of the result. In our performance experiments we used such polygons with increasing number of vertices, therefore the preprocessing and query time where proportional to the squared number of vertices.
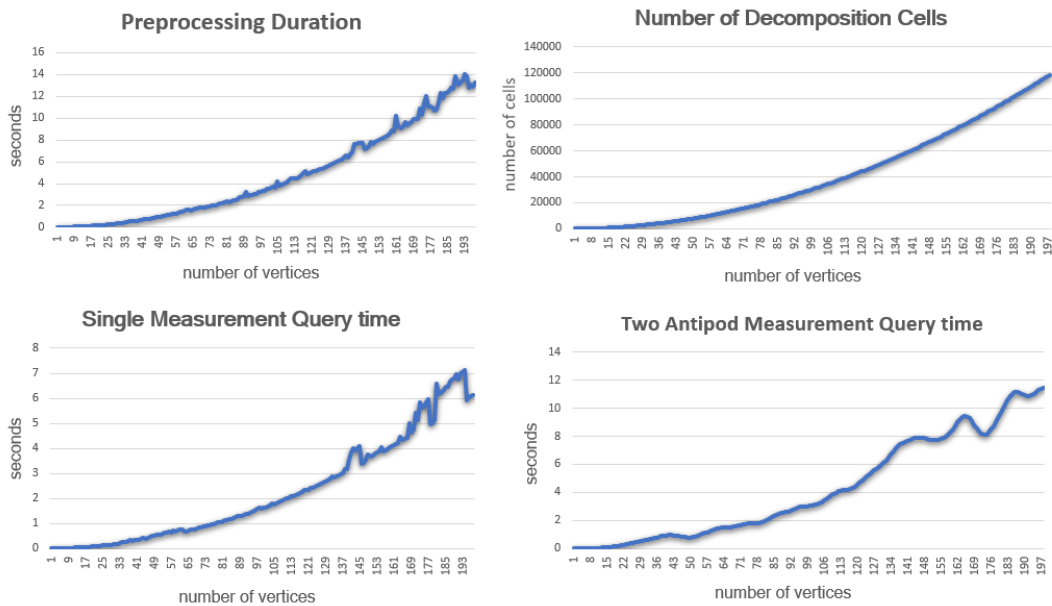
Figure 6.5:   From left to right, top to bottom. (1) The preprocessing time for a convex regular polygon with a radius of 1.0 and increasing number of vertices. (2) The number of three-dimensional RTD cells that were computed. The query time in these polygons is shown, both for a single measurement query of $d = 0.1$ at (3) and two antipodal measurements query of $d_1 = d_2 = 0.5$ at (4). The preprocessing time, the number of cells and the running time for both types of queries are behaving asymptotically as (*number of vertices*)^2, as expected in a perfect polygon.

33

Chapter 7

Discussion and Future Work

We have presented data structures to retrieve the unknown pose of a sensor in a known polygonal environment, in an output-sensitive fashion, based on depth measurements. The data structure for a single depth measurement provides two-dimensional surface patches as answers, and the data structure for two-antipodal measurements provides elliptical arcs as answers. The obvious next goal is to add one or more measurements from the same position in the workspace to reduce the answers to a finite number of poses. Figure 7.1 suggests an easy way to exploit our antipodal-measurement results to reduce the set of potential poses to a finite set of points, albeit no longer in an output-sensitive manner.
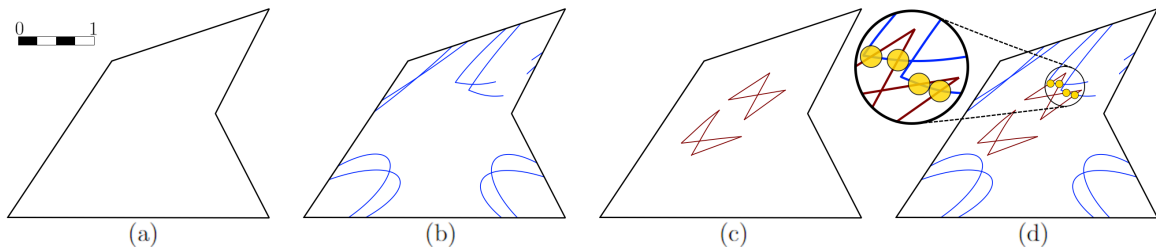

Figure 7.1: Given a workspace (a), if a sensor measured two measurements $d_1 = 0.6, d_2 = 0.8$ in antipodal directions, its possible locations are shown in (b). On the other hand, if the measurements values were $d_1 = 1.4, d_2 = 2.2$, its possible locations are shown at (c). If we know a sensor measured the first two antipodal measurements, then rotated to some unknown direction, and then measured the second pair of antipodal measurements, the possible locations of the sensor are the intersection of (b) and (c), shown in (d). Assuming general position, there will be a finite set of points the sensor might be in, marked by yellow discs in (d).

Additionally, we wish to allow movement of the sensor (robot) between measurements. Extending the exact analysis of the type we have applied in this thesis to more measurements or to allowing movement between measurements seems rather hard. We remark that our analysis here has already been used in a companion paper [3], which takes a numerical approach to the problem, in a system that makes three or more measurements, and using numerical methods well approximates the pose of the sensor, both in simulation and with a physical robot.

The data structures described in the thesis are intended to be computed once and queried many times, but in other use cases a workspace may be queried only once. What is the best time complexity required to answer a single query without any preprocessing? Another variant to consider is one in which dynamic modifications of the workspace are allowed in the form of removing a vertex and its edges and inserting a new edge connecting its adjacent vertices, or adding a new vertex connected with two new edges to an existing edge endpoints as a replacement for it. Can one construct dynamic data structures that can be updated efficiently under these modifications, and answer queries in an output-sensitive manner? Finally a major open issue is to make the algorithm robust to imprecision in the workspace modelling or the sensor readings.

Appendix A

Plane Projection of Measurement Results

A.1 Plane Projection Curves of a Single Measurement Result

Given a sensor reading d, in Section 3.2, we discussed the shape of the projection of all the configurations inside a single cell C of the RTD that yield the reading d. In this section we analyze the projected curved p_l^C, p_r^C, which are circular arcs or conchoids of Nicomedes; see Section 3.2 for details. Analytical description of the arcs is trivial, therefore we will focus on the description of the conchoids. Given a cell $C(e_t^C, e_b^C, v_l^C, v_r^C, \theta^C)$, assume, w.l.o.g., that the top and bottom edges are not vertical, and denote by $e_t^C(x) = x m_{e_t} + b_{e_t}$ the equation of the line supporting the top edge and analogously for the bottom edge $e_b^C(x) = x m_{e_b} + b_{e_b}$.

The conchoid of Nicomedes is driven from a fixed point q, a straight line ℓ and a length d, where for every line through q that intersects ℓ, the two points on the line which are at distance d from the intersection are on the conchoid. In our case, q is the limiting vertex, the straight line is the line supporting the top edge of the cell and d is the query measurement.

The classic equation for a conchoid, assuming that $q = (0,0)$ and the line ℓ is $x = a$, is given by:

$$(x - a)^2 (x^2 + y^2) = d^2 x^2 \ .$$

For the more general case, where $q = (x_0, y_0)$ for some x_0, y_0 and the line ℓ makes angle t with the positive x-axis and is at a distance a from q, is given by the following:

$$s = \sin t \ , \quad c = \cos t \ ,$$

$$
\begin{aligned}
((x - x_0)c - (y - y_0)s - a)^2 &(((x - x_0)c - (y - y_0)s)^2 + \\
((x - x_0)s + (y - y_0)c)^2) &= d^2((x - x_0)c - (y - y_0)s)^2
\end{aligned}
\tag{A.1}
$$

In our case, ℓ supports the top edge, and its angle is defined by its slope:

$$t = \tan^{-1}(m_{e_t}), \quad \sin\tan^{-1}(m_{e_t}) = \frac{m_{e_t}}{\sqrt{1 + m_{e_t}^2}}, \quad \cos\tan^{-1}(m_{e_t}) = \frac{1}{\sqrt{1 + m_{e_t}^2}} .$$

By using the above identities in Equation (A.1), we get:

$$\left((x - x_0)\frac{m_{e_t}}{\sqrt{1 + m_{e_t}^2}} - (y - y_0)\frac{1}{\sqrt{1 + m_{e_t}^2}} - a \right)^2$$

$$\cdot \left(\left((x - x_0)\frac{m_{e_t}}{\sqrt{1 + m_{e_t}^2}} - (y - y_0)\frac{1}{\sqrt{1 + m_{e_t}^2}} \right)^2 \right.$$

$$\left. + \left((x - x_0)\frac{1}{\sqrt{1 + m_{e_t}^2}} + (y - y_0)\frac{m_{e_t}}{\sqrt{1 + m_{e_t}^2}} \right)^2 \right) =$$

$$d^2 \left((x - x_0)\frac{m_{e_t}}{\sqrt{1 + m_{e_t}^2}} - (y - y_0)\frac{1}{\sqrt{1 + m_{e_t}^2}} \right)^2 ,$$

$$\frac{1}{1 + m_{e_t}^2}\left((x - x_0)\frac{m_{e_t}}{\sqrt{1 + m_{e_t}^2}} - (y - y_0)\frac{1}{\sqrt{1 + m_{e_t}^2}} - a \right)^2 \left((m_{e_t}(x - x_0) - (y - y_0))^2 \right.$$

$$\left. + ((x - x_0) + m_{e_t}(y - y_0))^2 \right) = \frac{1}{1 + m_{e_t}^2}d^2 (m_{e_t}(x - x_0) - (y - y_0))^2 ,$$

$$\left(\frac{m_{e_t}(x - x_0) - (y - y_0)}{\sqrt{1 + m_{e_t}^2}} - a \right)^2 \left(m_{e_t}^2(x - x_0)^2 - 2m_{e_t}(x - x_0)(y - y_0) + (y - y_0)^2 \right.$$

$$\left. + (x - x_0)^2 + 2m_{e_t}(x - x_0)(y - y_0) + m_{e_t}^2(y - y_0)^2 \right) = d^2 (m_{e_t}(x - x_0) - (y - y_0))^2 ,$$

$$\left(\frac{m_{e_t}(x - x_0) - (y - y_0)}{\sqrt{1 + m_{e_t}^2}} - a \right)^2 \left((x - x_0)^2 + (y - y_0)^2 \right)\left(1 + m_{e_t}^2 \right) =$$

$$d^2 (m_{e_t}(x - x_0) - (y - y_0))^2 ,$$

$$\left(m_{e_t}(x - x_0) - (y - y_0) \pm a\sqrt{1 + m_{e_t}^2} \right)^2 \left((x - x_0)^2 + (y - y_0)^2 \right) =$$

$$d^2 (m_{e_t}(x - x_0) - (y - y_0))^2 .$$

The term with $\pm$ sign is determined by the relation between q and ℓ. If ℓ is above q, we use the plus sign, and if ℓ is below q, we use the minus sign. In summary, the equation that represent the curve p_l^C within a cell is (can be defined analogously for p_r^C):

$$\left(m_{e_t}(x - x_{v_l}) - (y - y_{v_l}) \pm a\sqrt{1 + m_{e_t}^2} \right)^2 \left((x - x_{v_l})^2 + (y - y_{v_l})^2 \right)$$

$$= d^2 (m_{e_t}(x - x_{v_l}) - (y - y_{v_l}))^2 .$$

37

The Overall Region of Potential Sensor Positions for a Fixed Reading: Given a sensor reading d, we retrieve all the cells with non-empty potential poses using the sorted list of the RTD-based data structure. For each such cell C we compute the region $\Psi_d(C)$.

Alternatively, we may wish to return the (two-dimensional) union of the regions $\Psi_d(C)$, for all the cells that we retrieved. See Figure 6.1 for an illustration. Each region $\Psi_d(C)$ has a constant descriptive complexity, and if we retrieved k such regions, their union can be easily computed in time $O((k+m)\log(k+m))$, where m is the number of pairs of regions that intersect. In the worst case $m = O(k^2)$. It might be the case that special properties of the regions $\Psi_d(C)$ and their juxtaposition could be used to show that the complexity of the union is $o(k^2)$. We leave this as an open problem for further research.

A.2 Plane Projection curve of Antipodal Measurements Result

Given two measurements d_1, d_2, and a fixed RTD cell, all positions of a sensor that can measure d_1 at some angle θ_1 and d_2 at angle $\theta_2 = \theta_1 + \pi$, measuring the distance to the top and bottom edges of the cell respectively, can be calculated analytically. Consider all the line segments of length $d_1 + d_2$ with one endpoint on the top and the other on the bottom edge. Each possible position of the sensor can be expressed as a point placed on one of these segments, at distance d_1 from the endpoint that lies on the top edge. The curve that is the union of all these points, is a Glissette [2], which in our special case is an ellipse.

Glissettes are the curves traced out by a point on a curve γ, where γ slides along two other fixed curves. In our case specifically, γ is a fixed line segment s, sliding while each of its endpoints is in contact with one line (the two lines are not necessary orthogonal to one another), and the Glissette is defined as the trace of an arbitrary point on s (not necessarily the mid-point). A solution for the case where the two fixed lines are at right angle is given at [2, p. 51]. We solve it here for the general case.

For simplicity, assume that one of the fixed lines is the x-axis, it intersect the other fixed line at the origin, and the angle between them is α. If the moving line segment has length $d_1 + d_2$, and the designated point to be traced is at distance d_1, d_2 from the segment's endpoints, the Glissette points satisfy the following, for any angle ϕ in **oblique** coordinates:

$$x \sin \alpha = f(\phi), \quad y \sin \alpha = f(\phi + \alpha).$$

Where $f(\cdot)$ is the **tangential polar equation**. In our case:

$$f(\phi) = \begin{cases} d_1 \cos \phi, & \text{if } \phi \in [0, \pi/2], \\ d_2 \cos \phi, & \text{else} \end{cases}.$$

Again, these expressions represent the oblique coordinates. When transforming into Cartesian coordinates, we get:

$$y = d_2 \cos(\phi + \alpha), \quad x = \frac{d_1 \cos \phi}{\sin \alpha} + \frac{y}{\tan \alpha}.$$

The above expressions are correct for any angle ϕ, and we would like to express all the result points in a single equation. Next, we perform angle elimination:

$$\theta = \arccos\left(\frac{y}{d_2}\right) - \alpha \,,$$

$$x = \frac{d_1 \cos\left(\arccos\left(\frac{y}{d_2}\right) - \alpha\right)}{\sin\alpha} + \frac{y}{\tan\alpha} \,,$$

$$x = \frac{d_1\left(\cos\arccos\frac{y}{d_2}\cos\alpha + \sin\arccos\frac{y}{d_2}\sin\alpha\right)}{\sin\alpha} + \frac{y}{\tan\alpha} \,,$$

$$x = \frac{d_1\left(\frac{y}{d_2}\cos\alpha \pm \sqrt{1 - \frac{y^2}{d_2^2}}\sin\alpha\right)}{\sin\alpha} + \frac{y}{\tan\alpha} \,,$$

$$x = d_1\left(\frac{y}{d_2}\frac{\cos\alpha}{\sin\alpha} \pm \sqrt{1 - \frac{y^2}{d_2^2}}\right) + \frac{y}{\tan\alpha} \,,$$

$$x = \frac{d_1}{d_2}\frac{y}{\tan\alpha} \pm d_1\sqrt{1 - \frac{y^2}{d_2^2}} + \frac{y}{\tan\alpha} \,,$$

$$x - \left(1 + \frac{d_1}{d_2}\right)\frac{y}{\tan\alpha} = \pm d_1\sqrt{1 - \frac{y^2}{d_2^2}} \,,$$

$$x^2 - 2x\left(1 + \frac{d_1}{d_2}\right)\frac{y}{\tan\alpha} + \left(1 + \frac{d_1}{d_2}\right)^2\frac{y^2}{\tan^2\alpha} = d_1^2\left(1 - \frac{y^2}{d_2^2}\right) \,,$$

$$x^2 - 2\frac{1 + \frac{d_1}{d_2}}{\tan\alpha}xy + \left(\frac{\left(1 + \frac{d_1}{d_2}\right)^2}{\tan^2\alpha} + \frac{d_1^2}{d_2^2}\right)y^2 = d_1^2 \,,$$

$$\frac{x^2}{d_1^2} - 2\frac{1 + \frac{d_1}{d_2}}{d_1^2\tan\alpha}xy + \left(\frac{\left(1 + \frac{d_1}{d_2}\right)^2}{d_1^2\tan^2\alpha} + \frac{1}{d_2^2}\right)y^2 = 1 \,. \tag{A.2}$$

The above equation already expresses all the points in the ellipse, but we would like to work with standard representation of rotated ellipses. Rotated ellipse equation centered at (x_0, y_0) with rotation angle θ:

$$\frac{((x - x_0)\cos\theta + (y - y_0)\sin\theta)^2}{a^2} + \frac{((x - x_0)\sin\theta - (y - y_0)\cos\theta)^2}{b^2} = 1 \,.$$

We assumed that the intersection is at the origin, therefore $x_0 = y_0 = 0$. Denote $\sin\theta = s, \cos\theta = c, \tan\theta = t$, then

$$\frac{(xc + ys)^2}{a^2} + \frac{(xs - yc)^2}{b^2} = 1 \,,$$

$$\left(\frac{c^2}{a^2} + \frac{s^2}{b^2}\right)x^2 + 2cs\left(\frac{1}{a^2} - \frac{1}{b^2}\right)xy + \left(\frac{s^2}{a^2} + \frac{c^2}{b^2}\right)y^2 = 1 \,.$$

From Equation (A.2) and the above we can derive three equations with three unknowns:

$$\frac{1}{d_1^2} = \frac{c^2}{a^2} + \frac{s^2}{b^2} \, , \tag{A.3}$$

$$-2\frac{1 + \frac{d_1}{d_2}}{d_1^2 \tan \alpha} = 2cs\left(\frac{1}{a^2} - \frac{1}{b^2}\right) , \tag{A.4}$$

$$\frac{(1 + \frac{d_1}{d_2})^2}{d_1^2 \tan^2 \alpha} + \frac{1}{d_2^2} = \frac{s^2}{a^2} + \frac{c^2}{b^2} \, . \tag{A.5}$$

We can solve these equations to calculate a, b, θ.
From Equation (A.3) it follows that

$$\frac{1}{d_1^2} = \frac{c^2}{a^2} + \frac{s^2}{b^2} = \frac{c^2 b^2 + s^2 a^2}{a^2 b^2} \, ,$$

$$a^2 b^2 = d_1^2 (c^2 b^2 + s^2 a^2) \, ,$$

$$a^2 = \frac{d_1^2 c^2 b^2}{b^2 - d_1^2 s^2} \, . \tag{A.6}$$

From Equation (A.4) it follows that

$$-2\frac{1 + \frac{d_1}{d_2}}{d_1^2 \tan \alpha} = 2cs\left(\frac{1}{a^2} - \frac{1}{b^2}\right) ,$$

$$\frac{1 + \frac{d_1}{d_2}}{csd_1^2 \tan \alpha} = \frac{1}{b^2} - \frac{1}{a^2} \, .$$

Assigning a^2 using Equation (A.6), we get

$$\frac{1 + \frac{d_1}{d_2}}{csd_1^2 \tan \alpha} = \frac{1}{b^2} - \frac{b^2 - d_1^2 s^2}{d_1^2 c^2 b^2} \, ,$$

$$\frac{1 + \frac{d_1}{d_2}}{csd_1^2 \tan \alpha} = \frac{d_1^2 c^2 - b^2 + d_1^2 s^2}{d_1^2 c^2 b^2} \, ,$$

$$\frac{1 + \frac{d_1}{d_2}}{s \tan \alpha} = \frac{d_1^2 - b^2}{cb^2} \, ,$$

$$\frac{c(1 + \frac{d_1}{d_2})}{s \tan \alpha} + 1 = \frac{d_1^2}{b^2} \, ,$$

$$b^2 = \frac{d_1^2}{\frac{c(1 + \frac{d_1}{d_2})}{s \tan \alpha} + 1} = \frac{d_1^2}{1 + \frac{c}{s \tan \alpha}(1 + \frac{d_1}{d_2})} \, . \tag{A.7}$$

Assigning b^2 using Equation (A.7) into Equation (A.6), we get:

$$a^2 = \frac{d_1^2 c^2 b^2}{b^2 - d_1^2 s^2} = \frac{d_1^2 c^2}{1 - \frac{d_1^2 s^2}{b^2}} = \frac{d_1^2 c^2}{1 - \frac{d_1^2 s^2}{\frac{d_1^2}{1 + \frac{c}{s\tan\alpha}(1 + \frac{d_1}{d_2})}}} \; ,$$

$$= \frac{d_1^2 c^2}{1 - s^2 - \frac{cs}{\tan\alpha}(1 + \frac{d_1}{d_2})} = \frac{d_1^2 c^2}{c^2 - \frac{cs}{\tan\alpha}(1 + \frac{d_1}{d_2})} = \frac{d_1^2}{1 - \frac{s}{c\tan\alpha}(1 + \frac{d_1}{d_2})} \; ,$$

$$a^2 = \frac{d_1^2}{1 - \frac{s}{c\tan\alpha}(1 + \frac{d_1}{d_2})} \; . \tag{A.8}$$

We can now return to Equation (A.4), and replace a^2 and b^2 with the expressions obtained in Equation (A.8) and Equation (A.7) respectively:

$$\frac{(1 + \frac{d_1}{d_2})^2}{d_1^2 \tan^2\alpha} + \frac{1}{d_2^2} = \frac{s^2}{a^2} + \frac{c^2}{b^2} \; ,$$

$$\frac{(1 + \frac{d_1}{d_2})^2}{d_1^2 \tan^2\alpha} + \frac{1}{d_2^2} = \frac{s^2}{\frac{d_1^2}{1 - \frac{s}{c\tan\alpha}(1 + \frac{d_1}{d_2})}} + \frac{c^2}{\frac{d_1^2}{1 + \frac{c}{s\tan\alpha}(1 + \frac{d_1}{d_2})}} \; ,$$

$$\frac{(1 + \frac{d_1}{d_2})^2}{\tan^2\alpha} + \frac{d_1^2}{d_2^2} = s^2(1 - \frac{s}{c\tan\alpha}(1 + \frac{d_1}{d_2})) + c^2(1 + \frac{c}{s\tan\alpha}(1 + \frac{d_1}{d_2})) \; ,$$

$$\frac{(1 + \frac{d_1}{d_2})^2}{\tan^2\alpha} + \frac{d_1^2}{d_2^2} = s^2 - \frac{ts^2}{\tan\alpha}(1 + \frac{d_1}{d_2}) + c^2 + \frac{c^2}{t\tan\alpha}(1 + \frac{d_1}{d_2}) \; ,$$

$$\frac{(1 + \frac{d_1}{d_2})^2}{\tan^2\alpha} + \frac{d_1^2}{d_2^2} = 1 + \frac{c^2 - t^2 s^2}{t\tan\alpha}(1 + \frac{d_1}{d_2}) \; ,$$

$$\frac{(1 + \frac{d_1}{d_2})^2}{\tan^2\alpha} + \frac{d_1^2}{d_2^2} - 1 = \frac{1 - 2s^2}{tc^2 \tan\alpha}(1 + \frac{d_1}{d_2}) \; ,$$

$$\frac{(1 + \frac{d_1}{d_2})^2}{\tan^2\alpha} + (\frac{d_1}{d_2} - 1)(\frac{d_1}{d_2} + 1) = \frac{1 - 2s^2}{tc^2 \tan\alpha}(1 + \frac{d_1}{d_2}) \; ,$$

$$\frac{(1 + \frac{d_1}{d_2})}{\tan^2\alpha} + \frac{d_1}{d_2} - 1 = \frac{1 - 2s^2}{tc^2 \tan\alpha} \; ,$$

$$\frac{(1 + \frac{d_1}{d_2})}{\tan\alpha} + \frac{d_1 \tan\alpha}{d_2} - \tan\alpha = \frac{1 - 2s^2}{tc^2} \; ,$$

$$\frac{d_1 + d_2 + d_1 \tan^2\alpha - d_2 \tan^2\alpha}{d_2 \tan\alpha} = \frac{1 - 2s^2}{tc^2} \; ,$$

$$\frac{d_1(1 + \tan^2\alpha) + d_2(1 - \tan^2\alpha)}{d_2 \tan\alpha} = \frac{1 - 2s^2}{tc^2} \; ,$$

41

$$\frac{1 - 2s^2}{c^2 t} = \frac{cos(2\theta)}{c^2 t} = 2\cot(2\theta) \ ,$$

$$1 + \tan^2 \alpha = \frac{1}{\cos^2 \alpha}, \quad 1 - \tan^2 \alpha = \frac{1 - 2\sin^2 \alpha}{\cos^2 \alpha} = \frac{\cos(2\alpha)}{\cos^2 \alpha} \ ,$$

$$\frac{\frac{d_1}{\cos^2 \alpha} + \frac{d_2 \cos(2\alpha)}{\cos^2 \alpha}}{d_2 \tan \alpha} = 2\cot(2\theta) \ ,$$

$$\frac{d_1 + d_2 \cos(2\alpha)}{d_2 \sin \alpha \cos \alpha} = 2\cot(2\theta) \ ,$$

$$\frac{\frac{d_1}{d_2} + \cos(2\alpha)}{\sin(2\alpha)} = \cot(2\theta) \ ,$$

$$\theta = \frac{1}{2}\cot^{-1}\left(\frac{\frac{d_1}{d_2} + \cos(2\alpha)}{\sin(2\alpha)}\right) \ . \tag{A.9}$$

With θ calculated, we can go back to Equation (A.8) and Equation (A.7) and easily calculate their exact value. We have calculate the fixed values a, b, θ under the assumption that the line supporting the bottom edge is the x-axis and it intersects the line supporting the top edge at the origin. The ellipse is expressed by the equation:

$$\frac{(x\cos\theta + y\sin\theta)^2}{a^2} + \frac{(x\sin\theta - y\cos\theta)^2}{b^2} = 1 \ .$$

In full generality, for any cell, we use $\theta' = \theta + \tan^{-1}(m_b^C)$, and offset the center of the ellipse to the intersection of the lines supporting the top and bottom edge of the cell (x_0, y_0):

$$\frac{((x - x_0)\cos\theta' + (y - y_0)\sin\theta')^2}{a^2} + \frac{((x - x_0)\sin\theta' - (y - y_0)\cos\theta')^2}{b^2} = 1 \ .$$

Appendix B

Combinatorial Result Change Handling

In this section, we provide more details on the analysis in Section 4.2. All the tuples with the same first parameter e, sorted in increasing d value, are the events of the spatial sweep with the surface $\Sigma(e, d)$. We compute the result $\sigma(e, d)$ for infinitesimal values of d, by considering only e and its two adjacent edges. From this points on, result edges will be inserted or removed only when handling combinatorial change tuples. We handle each combinatorial change (e, θ, d, U) according to the event type. We fully describe how to handle a change (e, θ, d, U) of type I and V, as types II, III and IV are analog to I. Let us first introduce some definitions and general notes:

(a) We say that a result edge η' is defined by e and a vertex v when η' is the union of configurations where a rod of length d is sliding on e and v, and we denote this rod by $s(e, v)$.

(b) We say that a result edge η'' is defined by e and another edge e_1 when η'' is the union of configurations where a rod of length d is sliding with one endpoint on e and another endpoint on e_1, and we denote this rod by $s(e, e_1)$.

(c) The limits α, β of the angle range $[\alpha, \beta]$ throughout which a result edge prevails, can be specified by a fixed angle each, or by the labels of the workspace features that induced these angles for a specific d value.

(d) Note that we refer to 'the' segment (or 'the' rod) of length d with its endpoints on some vertex and e, while there may be two such segments. The uniqueness of the segment is specified by the constraint on its angle, for example $\delta \leqslant \theta$ in the first sub-case of type I handling.

(e) For the simplicity of exposition, we assume that e is parallel to the x-axis.

(f) We say that a vertex is a *reflex vertex relative to a line (resp. segment)* if both of its incident edges are in the same half-plane induced by the line (resp. the line supporting the segment).

(g) By 'the angle of a segment/rod' we refer to the angle that the line supporting the segment/rod creates with the positive direction of the x-axis.

Combinatorial change of type I : d is the minimum value in which a result vertex of type (i) involving two vertices v_1, v_2 exists (namely an intersection of two $\chi_e(\cdot)$ surfaces intersect $\Sigma(e, d)$), where v_1 is the vertex closer to e. We use the following assumptions and definitions:

(a) We assume that v_2 is a reflex vertex relative to $\overline{v_1 v_2}$; otherwise, d is also the maximum value in which such a result vertex exists, and we will handle it as a change of type II (see Figure 4.4(a) for such a scenario).

(b) The vertex v_1 is clearly a reflex vertex relative to $\overline{v_1 v_2}$, and we will assume that the edges incident to v_2 are on the left halfplane relative to the direction $\overrightarrow{v_2 v_1}$.

(c) Let e_1 be the edge of v_2 visible from v_1.

The handling operations depend on two predicates: whether the edges of both v_1, v_2 are in the same halfplane relative to $\overline{v_1 v_2}$ or not, and whether a result edge defined by e, e_1 exists or not in $\sigma(e, d')$ for d' infinitesimally smaller than d. The sub-cases are illustrated in Figure 4.3.

- A result edge η_1 defined by e, e_1 exists in the angle range $[\alpha, \gamma]$ for some α, where γ depends on d and it is defined as the angle that $s(e, e_1)$ (that also passes through v_1 at the time of handling the event) makes with the x-axis. There must be another result edge η_2 defined by e, v_1 in the angle range $[\gamma, \beta]$ for some β.
  - If the edges of both v_1, v_2 are in the same halfplane relative to $\overline{v_1 v_2}$:
    * Find and remove η_1 and η_2.
    * Add a result edge defined by e, v_2 in the angle range $[\delta, \theta]$ where $\delta \leqslant \theta$ is the angle of the rod $s(e, v_2)$ (there are two such rods and we choose one of them by the constraint $\delta \leqslant \theta$). Recall that θ is a parameter of the combinatorial change tuple.
    * Add a result edge defined by e, v_1 in the angle range $[\theta, \beta]$.
  - If the edges of v_1, v_2 are not in the same halfplane relative to $\overline{v_1 v_2}$:
    * Find and remove η_1 and η_2.
    * Add a result edge defined by e, v_2 in the angle range $[\theta, \delta]$ where $\delta \geqslant \theta$ is the angle of the rod $s(e, v_2)$.
    * Add a result edge defined by e, v_1 in the angle range $[\theta, \beta]$.
- A result edge defined by e, e_1 does not exists. Therefore, there must be a result edge η_3 defined in the angle range $[\alpha, \beta]$ where $\theta \in (\alpha, \beta)$.
  - If the edges of both v_1, v_2 are in the same halfplane relative to $\overline{v_1 v_2}$:
    * Find and remove η_3.
    * Add a result edge defined by e, e_1 in the angle range $[\alpha, \phi]$ where $\phi \geqslant \alpha$ is the angle of the rod $s(e, e_1)$ (that also passes through v_1 at the time of handling the event).

* Add a result edge defined by e, v_2 in the angle range $[\delta, \theta]$ where $\delta \leqslant \theta$ is the angle of the rod $s(e < v_2)$.
    * Add a result edge defined by e, v_1 in the angle range $[\theta, \beta]$.
- If the edges of v_1, v_2 are not in the same halfplane relative to $\overline{v_1 v_2}$:
    - Find and remove η_3.
    - Add a result edge defined by e, e_1 in the angle range $[\alpha, \phi]$ where $\phi \geqslant \alpha$ is the angle of the rod $s(e, e1)$ (that also passes through v_1 at the time of handling the event).
    - Add a result edge defined by e, v_2 in the angle range $[\theta, \delta]$ where $\delta \geqslant \theta$ is the angle of the rod $s(e, v_2)$.
    - Add a result edge defined by e, v_1 in the angle range $[\theta, \beta]$.

Combinatorial change of type V : d is the length of the shortest line segment, denoted $\bar{s}$, which is contained in the workspace, from a vertex v to e. We use the following assumptions and definitions:

(a) Let e_1, e_2 be the edges incident to v, where e_1 is met first when we go counterclockwise around the vertex v from $\bar{s}$.

(b) Let θ be the angle create by the segment $\bar{s}$ and the positive x-axis.

(c) We assume that e_2 is in the right halfplane relative to $\bar{s}$, where $\bar{s}$ is directed from v to its endpoint on e. The case where e_2 is in the left halfplane can be handled symmetrically.

(d) Let $\gamma \leqslant \theta$ and $\phi \geqslant \theta$ be the angles of the two rods $s(e, v)$ (while typically we refer to a single rod, as we have some constraint on the rod orientation, in this case we consider both rods.

The handling operations of the change depend on whether there are already result edges defined by e_1 and e_2:

- There are result edges defined by e, e_1 and e, e_2 (Figure B.1(a)), denoted by η_1 and η_2 respectively:
  Let $[\alpha_1, \beta_1], [\alpha_2, \beta_2]$ be the angle ranges η_1, η_2 exist in, respectively.
    - Find and remove η_1 and η_2.
    - Add a result edge defined by e, e_1 in the angle range $[\alpha_1, \gamma]$.
    - Add a result edge defined by e, e_1 in the angle range $[\phi, \beta_1]$.
    - Add a result edge defined by e, e_2 in the angle range $[\alpha_2, \gamma]$.
    - Add a result edge defined by e, e_2 in the angle range $[\phi, \beta_2]$.
- There is a single result edge defined by either e, e_1 or e, e_2:
  Assume that only the result edge defined by e, e_2 exists, and denote it by η_2. The case where only the result edge defined by e, e_1 exists is handled symmetrically. First, η_2 should be removed from R. The result edges that should be added depends on these sub cases:

– Both e_1, e_2 are in the right halfplane relative to $\bar{s}$ (Figure B.1(b)):
  * A result edge defined by e, v that exists in the angle range $[\gamma, \phi]$.
  * Add a result edge defined by e, e_2 in the angle range $[\alpha_2, \gamma]$.
  * Add a result edge defined by e, e_2 in the angle range $[\phi, \beta_2]$.
– The edges e_1, e_2 are in different halfplanes relative to $\bar{s}$ (Figure B.1(c)):
  * A result edge defined by e, e_1 that exists in the angle range $[\gamma, \phi]$.
  * Add a result edge defined by e, e_2 in the angle range $[\alpha_2, \gamma]$.
  * Add a result edge defined by e, e_2 in the angle range $[\phi, \beta_2]$.

• There are no result edges defined by e, e_1 or e, e_2:
  A new hole in the result should be added, with edges depending on the following sub-cases:

  – Both e_1, e_2 are in the right halfplane relative to $\bar{s}$ (Figure B.1(d)):
    * A result edge defined by e, v that exists in the angle range $[\gamma, \phi]$.
    * A result edge defined by e, e_2 that exists in the angle range $[\gamma, \phi]$.
  – The edges e_1, e_2 are in different halfplanes relative to $\bar{s}$ (Figure B.1(e)):
    * A result edge defined by e, e_1 that exists in the angle range $[\gamma, \phi]$.
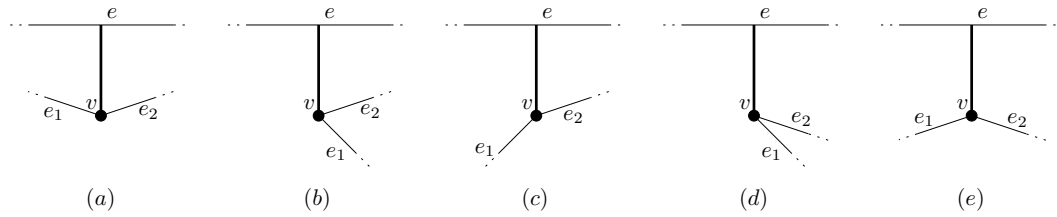   * A result edge defined by e, e_2 that exists in the angle range $[\gamma, \phi]$.

Figure B.1: Examples of the sub-cases of handling combinatorial change of type (V). The bold line is a segment of length d. There is a result edge defined by e, e_1 fof length $< d$ in (a). There is a result edge defined by e, e_2 of length $< d$ in (a,b,c). Both e_1 and e_2 are in the same halfplane relative to the shortest line between v and e, in (b)(d).

Bibliography

[1] Boris Aronov, Leonidas J. Guibas, Marek Teichmann, and Li Zhang. Visibility queries and maintenance in simple polygons. *Discret. Comput. Geom.*, 27(4):461–483, 2002.

[2] William Henry Besant. *Notes on Roulettes and Glissettes*. Deighton, Bell, Cambridge, 1870.

[3] Michael M. Bilevich, Steven M. LaValle, and Dan Halperin. Sensor localization by few distance measurements via the intersection of implicit manifolds. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, United Kingdom, May 28 - June 3, 2023*. IEEE, 2023.

[4] Prosenjit Bose, Anna Lubiw, and J. Ian Munro. Efficient visibility queries in simple polygons. *Comput. Geom.*, 23(3):313–335, 2002.

[5] Wolfram Burgard, Dieter Fox, Daniel Hennig, and Timo Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In William J. Clancey and Daniel S. Weld, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2*, pages 896–901. AAAI Press / The MIT Press, 1996.

[6] Jinwoo Choi, Minyong Choi, Il Hong Suh, and Wan Kyun Chung. Topological localization using sonar gridmap matching in home environment. In *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*, pages 4910–4915. IEEE, 2010.

[7] Mark de Berg, Leonidas J. Guibas, and Dan Halperin. Vertical decompositions for triangles in 3-space. *Discret. Comput. Geom.*, 15(1):35–61, 1996.

[8] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008.

[9] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, USA, May 10-15, 1999, Proceedings*, pages 1322–1328. IEEE Robotics and Automation Society, 1999.

[10] Gamini Dissanayake, Paul M. Newman, Steve Clark, Hugh F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Trans. Robotics Autom.*, 17(3):229–241, 2001.

[11] Gregory Dudek, Kathleen Romanik, and Sue Whitesides. Localizing a robot with minimum travel. *SIAM J. Comput.*, 27(2):583–604, 1998.

[12] Hugh F. Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: Part I. *IEEE Robotics Autom. Mag.*, 13(2):99–110, 2006.

[13] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[14] Alberto Elfes. Sonar-based real-world mapping and navigation. In Ingemar J. Cox and Gordon T. Wilfong, editors, *Autonomous Robot Vehicles*, pages 233–249. Springer, 1990.

[15] Sean P. Engelson and Drew V. McDermott. Error correction in mobile robot map learning. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation, Nice, France, May 12-14, 1992*, pages 2555–2560. IEEE Computer Society, 1992.

[16] Efi Fogel, Dan Halperin, and Ron Wein. *CGAL Arrangements and Their Applications - A Step-by-Step Guide*, volume 7 of *Geometry and Computing*. Springer, 2012.

[17] Dieter Fox, Sebastian Thrun, Wolfram Burgard, and Frank Dellaert. Particle filters for mobile robot localization. In Arnaud Doucet, Nando de Freitas, and Neil J. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, pages 401–428. Springer, 2001.

[18] Subir Kumar Ghosh and David M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM J. Comput.*, 20(5):888–910, 1991.

[19] Nir Goren, Efi Fogel, and Dan Halperin. CGAL made more accessible. *CoRR*, abs/2202.13889, 2022.

[20] Leonidas J. Guibas, Rajeev Motwani, and Prabhakar Raghavan. The robot localization problem. *SIAM J. Comput.*, 26(4):1120–1138, 1997.

[21] José E. Guivant and Eduardo Mario Nebot. Optimization of the simultaneous localization and map-building algorithm for real-time implementation. *IEEE Trans. Robotics Autom.*, 17(3):242–257, 2001.

[22] Dan Halperin and Micha Sharir. Arrangements. In J. E. Goodman, J. O'Rourke, and Cs. D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, chapter 28, pages 723–762. Chapman & Hall/CRC, Boca Raton, FL, 3rd edition, 2018.

[23] Patric Jensfelt and Steen Kristensen. Active global localization for a mobile robot using multiple hypothesis tracking. *IEEE Trans. Robotics Autom.*, 17(5):748–760, 2001.

[24] Rudolf E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960.

[25] Sven Koenig, Joseph S. B. Mitchell, Apurva Mudgal, and Craig Tovey. A near-tight approximation algorithm for the robot localization problem. *SIAM Journal on Computing*, 39(2):461–490, 2009.

[26] Philippe Moutarlier and Raja Chatila. An experimental system for incremental environment modelling by an autonomous mobile robot. In Vincent Hayward and Oussama Khatib, editors, *Experimental Robotics I, The First International Symposium, Montréal, Canada, June 19-21, 1989*, volume 139 of *Lecture Notes in Control and Information Sciences*, pages 327–346. Springer, 1989.

[27] Jason M. O'Kane and Steven M. LaValle. Localization with limited sensing. *IEEE Trans. Robotics*, 23(4):704–716, 2007.

[28] Micha Sharir and Shmuel Sifrony. Coordinated motion planning for two independent robots. *Ann. Math. Artif. Intell.*, 3(1):107–130, 1991.

[29] Randall Smith, Matthew Self, and Peter C. Cheeseman. Estimating uncertain spatial relationships in robotics. In Ingemar J. Cox and Gordon T. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer, 1990.

[30] Cyrill Stachniss, John J. Leonard, and Sebastian Thrun. Simultaneous localization and mapping. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics*, Springer Handbooks, pages 1153–1176. Springer, 2016.

[31] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.2 edition, 2023.

[32] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Auton. Robots*, 5(3-4):1–5, 1998.

[33] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Intelligent robotics and autonomous agents. MIT Press, 2005.

[34] Alireza Zarei and Mohammad Ghodsi. Query point visibility computation in polygons with holes. *Comput. Geom.*, 39(2):78–90, 2008.

תקציר

התיזה עוסקת בבעיית איכון (localization) בפוליגון עם חורים במישור לרובוט עם חיישן מרחק תוך שימוש במדידה אחת או שתיים מנוגדות.

נעשתה עבודה רבה בתחום עבור רובוטים עם חיישנים מתקדמים שסורקים את סביבת העבודה ודוגמים אותו במספר רב של נקודות או דוגמים את הפוליגון הנראה בשלמותו. התזה מחדשת ונותנת פתרונות עבור רובוטים עם חיישנים מוגבלים ומאפשרת הוזלה בעלות הרובוט, או הקטנת התקשורת הדרושה כדי לשלוח את המידע שחיישני הרובוט אוספים.

שני מבני נתונים המבוססים על פירוק סיבובי לטרפזים של מרחב המצבים האפשריים של החיישן, שהוא מרחב תלת ממדי (לציון מיקום וכיוון במישור), מוצגים, אשר משיגים זמן עיבוד מוקדם ומקום כגודל הפירוק כפול פקטור לוגריתמי, ומאפשרים לענות על שאילתה בזמן אופטימלי ביחס לפירוק בהוספה של מספר לוגריתמי של פעולות בגודל סביבת העבודה. שני מבני הנתונים יחסית פשוטים, ומימשנו אותם בעזרת ספריית CGAL. למרות ששני מבני הנתונים משיגים זמן שאילתה אופטימלי ביחס לפירוק, הם לא משיגים זמן שאילתה אופטימלי ביחס למספר המינימלי של הקודקודים וקשתות הנדרשים לתאר את התוצאה, שהוא זמן השאילתה האופטימלי.

לאחר מכן אנחנו מציעים שני מבני נתונים קצת יותר סבוכים שמשיגים זמן עיבוד מוקדם ומקום זהה למבני הנתונים הפשוטים, ומצליחים להשיג זמן שאילתה אופטימלי (עד פקטור לוגריתמי בגודל סביבת העבודה), כלומר כמספר המינימלי של קודקודים וקשתות הנדרשים כדי להציג את התוצאה. מבני הנתונים עבור מדידה אחת עוקב אחרי השינויים הקומבינטורים של התוצאה עבור כל ערך מדידה, ושומר בצורה יעילה את כל התוצאות האפשריות לשאילתה. מבנה הנתונים עבור שתי מדידות משתמש בתאים של הפירוק הסיבובי לטרפזים, אך הוא מאחד תאים שעבורם הפירוק לא נחוץ עבור ערכי דגימה מסויימים.

לבסוף, אנחנו מציגים את המימוש של שני מבני הנתונים הפשוטים יותר וכמה ניסויים שביצענו בעזרתם. אנחנו מעלים כמה שאלות פתוחות בנושא, כמו איך להתמודד עם חוסר דיוק של חיישן המדידה, האם אפשר להרחיב את השיטות שלנו ליותר משתי מדידות או לזוויות לא מנוגדות, ועוד.

הפקולטה למדעים מדויקים
ע״ש ריימונד ובברלי סאקלר
אוניברסיטת תל אביב
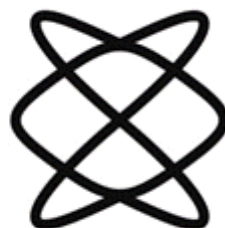
 בית הספר למדעי המחשב

איכון במישור בעזרת מדידת מרחק אחת או שתיים נגדיות

חיבור זה הוגש כחלק מהדרישות לקבלת התואר
"מוסמך אוניברסיטה" – .M.Sc באוניברסיטת תל אביב

על ידי
ברק עוגב

העבודה הוכנה בהדרכתו של
פרופ׳ דן הלפרין