Hindawi
Mobile Information Systems
Volume 2018, Article ID 5303616, 11 pages
https://doi.org/10.1155/2018/5303616

*Research Article*

# A Method to Incorporate Floor Plan Constraints into Indoor Location Tracking: A Voronoi Approach

**John D. Hobby**[1] **and Marzieh Dashti**[2]

[1]*Nokia Bell Labs, Murray Hill, NJ, USA*
[2]*Accenture The Dock, Ireland*

Correspondence should be addressed to Marzieh Dashti; marzieh.dashti@accenture.com

Received 12 January 2017; Revised 9 May 2017; Accepted 17 December 2017; Published 20 February 2018

Academic Editor: Olivier Julien



Indoor localization has attracted a lot of research effort in recent years due to the explosion of indoor location-based service (LBS) applications. Incorporating map constraints into localization algorithms reduces the uncertainty of walking trajectories and enhances location accuracy. Suitable maps for computer-aided localization algorithms are not readily available, and hence most researchers working on localization solutions manually create maps for their specific localization scenarios. This paper presents a method of generating indoor maps suitable for localization algorithms from CAD floor plans. Our solution is scalable for mass-market LBS deployment. We also propose an adapted map-filtering algorithm that utilizes map information extracted from CAD floor plans. We evaluate the performance of our solution via real-world Wi-Fi RF measurements.

## 1. Introduction

Accurate and robust indoor localization is a key enabler for numerous emergency and commercial services. Some examples of indoor location-based services (LBS) are as follows [1]: locate people on a map and navigate them to their destinations in shopping malls, airports, hospitals, and museums; recommend nearest business or services such as ATMs, retail stores, restaurants, and social events; produce location-based advertising; find and track stolen or lost objects; and monitor human activities. When displaying localization and tracking results, it is natural to superimpose the coordinates on the building floor plan, as an absolute location value is not of much use without its relation to the surrounding area map. In indoor LBS applications, the location data are visualized on a user-friendly building map displayed on the user device's screen. Besides the requirement of maps for location data visualization, map information can be utilized to enhance location accuracy and reduce uncertainty of walking trajectories.

Many indoor localization and tracking algorithms rely to a certain extent on map-based filtering methods to bound drift and noise-induced errors. These algorithms are most commonly based on particle filters [2]. The basic idea is fairly straightforward: the user's trajectory is described by a set of particles. The particle distribution models the measured trajectory as well as errors of the measurement systems [3]. Particles are not allowed to move to positions that violate the map constraints. For example, particles are not allowed to cross directly through walls. Particles that transit through such obstacles are down weighted or resampled [4].

Different ways of dealing with floor plan information are presented in the literature. In [5], particles crossing walls are eliminated by using a bitmap-based map matching algorithm. The state of the system and consequently the state of each particle consists of two pixel coordinates $x$ and $y$ on a bitmap. When the path of a particle (implemented as line painting on the bitmap) includes a nonwhite pixel, a collision with an obstacle has occurred and the weight of the particle is set to zero. Note that this requires a white representation of walkable space on the bitmap which includes, for example, the removal of doors or room names possibly depicted on the bitmap. In [6], the particle filter builds on a mixed graph/free space representation of indoor environments. While hallways, stair cases, and elevators are represented by edges in a graph, areas such as rooms are

represented by bounded polygons. Using this representation, both constrained motion such as moving down a hallway or going upstairs and less-constrained motion through rooms and open spaces are modeled. In [7], the authors assume that a floor plan is a set of walls, each described by the coordinates of the end points of the wall. Doors are modeled as gaps in the walls.

Generally speaking, the maps are basically drawings for human consumption, and they present some difficulties when used for algorithmic analysis. And, therefore, most researchers working on localization solutions manually create the required maps for their specific testing scenarios. While this approach is valid to test the performance of map-filtering algorithms, it cannot be scaled for high-volume commercial LBS applications.

As stated above and also argued in [3, 8], the requirements for the maps used for visualization and location estimation differ [8]. The lack of maps that are suitable for both, visualization and map-based filtering, is one of the main challenges for the mass-market deployment of indoor positioning systems. The building floor plan maps are commonly designed using computer-aided design (CAD) commercial software applications. Hence we aim at converting the floor plan CAD file in a format which is suitable as an input to the map-filtering algorithms. To the best of our knowledge, the only work that talks about this important problem is [3]. The authors present a parser that analyses standard CAD files to extract topological map information. This information is used to create an object-based map optimized for localization and tracking applications.

This paper presents a scalable method of generating indoor maps suitable for localization algorithms from CAD data. We discuss how to handle interior walls when using indoor tracking to locate smart phones or other devices based on technologies such as Wi-Fi signal strength. While we address the issues discussed in [3] with a new method, our focus is more on the need to get room polygons, which leads to planar subdivisions and the need to locate "doorway features." These are relatively narrow openings that should be treated like doorways. Another issue not discussed in [3] is that the building description contains more detail than is needed for tracking. We tried simplification strategies based on replacing polygonal lines that are well approximated by a single straight line segment. This speeds up the tracking algorithm with no significant effect on accuracy. We analyze the performance of our proposed algorithms using real-world RF fingerprint measurement data.

The remainder of this paper is structured as follows: in Section 2, we describe how to extract walls and room polygons' information from a floor plan CAD file. Section 3 describes our proposed tracking algorithm which incorporates floor plan constraints. Section 4 illustrates the tracking results using real-world Wi-Fi RF measurements taken in an enterprise building in Dublin. Section 5 concludes the paper.

## 2. Finding Walls and Room Polygons

There are at least two reasons for the location-finding algorithm itself to consider the floor plan and interior walls:

(1) Since people cannot walk through walls, it seems desirable for the tracking algorithm to forbid such trajectories.

(2) Depending on materials used, carrier frequency, and angle of incidence, walls tend to reflect a portion of the signal. Hence the tracking algorithm should expect walls to cause discontinuous changes in signal strength.

For Reason 2, it may suffice to have a set of line segments that the device being tracked is forbidden to cross. We extracted the line segments coordinates from our test bed building CAD floor plan and created an *edge list file*. However, Reason 2 suggests a dividing the building interior into a set of disjoint polygons, where some of the edges are designated as "doors" (in general, any part of a room boundary that the tracked device may pass through). We recorded the room polygons information into a *room polygon file*. In this section, we present in detail how to handle walls and room polygons.

Suppose the building floor plan is available in some machine-readable form (e.g., AutoCad®), and it is composed of *objects* such as "line," "polyline," "circular arc," and "text." Furthermore, it is divided into *layers*, and some layers may have multiple instances of "blocks." For example, the "furniture" layer may have "chair" blocks, where all chair blocks are affine transformations of each other.

The floor plans are basically drawings for human consumption, and they present some difficulties when used for algorithmic analysis. We have data for one floor of a $139 \times 101$ meter building in Dublin, Ireland. Our sample data had the following problems:

(1) Walls needed for room-based analysis are not on clearly defined layers; for example, some room dividers are in the furniture layer.

  (i) It helps to look at object names and block names as well as layer names.
  (ii) Objects named "Hatch" and blocks named "Door" are not walls.

(2) Heuristic tests are necessary to recognize features such as swinging doors and partitions in the furniture layer.

  (i) Instead of looking at rectangle aspect ratio, it is safer to compare the perimeter $P$ to the square root of the enclosed area $A$. For instance, $P/\sqrt{A} > 20/3$ implies a rectangle aspect ratio $> 9$, and it is meaningful for nonrectangles.

(3) Features such as door jambs and window frames provide numerous instances where different objects should share common $x$ or $y$ coordinates. These often do not match exactly.

  (i) Errors of this type are typically between 0.3 mm and 1 mm.

The object is to construct a set of polygons that accurately reflect the dimensions of the building (and its interior

Table 1: Parameters that control the door recognizer.

| Parameter | Value | Explanation |
|---|---|---|
| AuxDoorLim | 1 | 4th power of maximum-allowed aux door jamb error versus door length |
| MaybeAspect | 8 | Polyline furniture beyond this aspect ratio is a partition |
| DoorHrange | 80 | Swinging parts of a door must have handles in this interval |
| DoorH1range | 5 | Door rectangle sides must have handles in this interval |
| DJamAspect | 0.75 | Width versus length for excluded door jamb rectangle |
| DoorMinArc | Pi/4 | Minimum angle subtended by a door arc |
| DoorMaxArc | 7*Pi/12 | Maximum angle subtended by a door arc |
| DoorAspect | 6 | Minimum aspect ratio for a door rectangle |
| DoorWdErrFrac | 0.2 | Maximum door rectangle width error versus widest width |
| DoorWdErrvsL | 0.03 | Maximum door rectangle width error as fraction of length |
| DoorOffvsWd | 0.2 | Maximum door end squareness error distance versus widest width |
| DoorOffvsL | 0.03 | Maximum door end squareness error distance versus length |
| Bool Door1lineOK | True | May a door have one line rather than 2 parallel lines? |
| DoorShortArcvsL | 0.15 | Tolerance for door arc short of latch versus door length |
| DoorPastArcvsL | 0.20 | Tolerance for door arc past latch versus door length |
| DoorShortArcMin | 0.01 | Minimum of both tolerance for door arc short of latch versus door length |
| DoorPastArcMin | 0.05 | Minimum of both tolerance for door arc past latch versus door length |
| DoorMisfitArc | 0.03 | Fraction of door length by which door latch may misfit arc |
| DoorArcOffCtr | 0.05 | Fraction of door length for hinge versus arc center-squared error |
| DoorMinOffCtr | 0.026 | Minimum door length fractions for hinge versus squared arc center error |

walls) and show how it is possible to move from room to room. More precisely, we need a *planar subdivision*. A planar subdivision has vertices, edges, and faces, and the edges that meet at a common vertex are sorted by angle modulo 360 degrees. Furthermore, two faces meet at each edge, and it is possible to walk through the data structure in order to find the sequence of edges that describe any given face as a polygon [9]. For our application, the edges are polygonal lines, and certain segments are marked as doors (meaning that the tracking algorithm assumes its target can pass through).

Section 2.1 briefly outlines the heuristics for deciding which parts of the building description are walls and doorways. Then Section 2.2 explains how to convert the wall segments into a planar subdivision that defines rooms. Finally, Section 2.3 explains how to recognize places where doorway segments should be inserted so as to prevent large complicated regions from being erroneously treated as one room. This can be difficult to automate, so we also allow door segments to be added manually.

The generation of indoor maps from CAD data in the face of Problems 1-2 has been studied before in [3]. However, we focus more strongly on the need to get room polygons, and this leads to planar subdivisions (Section 2.2) and the door segment insertion problem (Section 2.3).

*2.1. Heuristics for Identifying Walls.* Heuristics are needed to recognize walls and doorways in the building description. To the extent that these could vary from one building to the next, the implementation needs to be as table-driven as possible; that is, customizing it to a new building should require new parameter values, not new algorithms. Although our sample data have "Door" blocks, only a small fraction of the doors are identified this way. Hence we have a Boolean expression for whether an object is worth keeping, and this is followed by logic that identifies doors. The door logic is controlled by numeric parameters as shown in Table 1.

In the tabular form of the Boolean decision, one table gives (layer name and decision index) pairs, and there are other such tables for object names and block names. The Boolean expression for whether to keep an object or treat it as junk is determined by another table where each node has a query value $q$ for one of the decision indices as well as "instructions" for $\leq q$ and $> q$. Each such "instruction" is either the index for another node in the table or a final decision ("Keep it," "Junk," or "Keep it if beyond MaybeAspect"). The main table has 6 of these nodes, and the table that gives decision indices for layer names considers 6 layer names and produces decision indices 0, 1, or 2.

Figure 1 exemplifies the input to the door-recognizing heuristics. The primary indication for the presence of a door is an elongated rectangle with a circular arc nearby. It is necessary to eliminate the picture of the swinging door and insert a segment marked "Door" that corresponds to the door in the closed position. Figure 1(a) has two horizontal lines that are near the desired closed-door segment. There are parameters in Table 1 that determine how to recognize such door jamb segments so that they can be removed.

*2.2. Building a Planar Subdivision.* The reason for wanting a planar subdivision is to have a complete set of room polygons with clearly defined geometry, where we can easily identify the two rooms connected by each doorway. Since

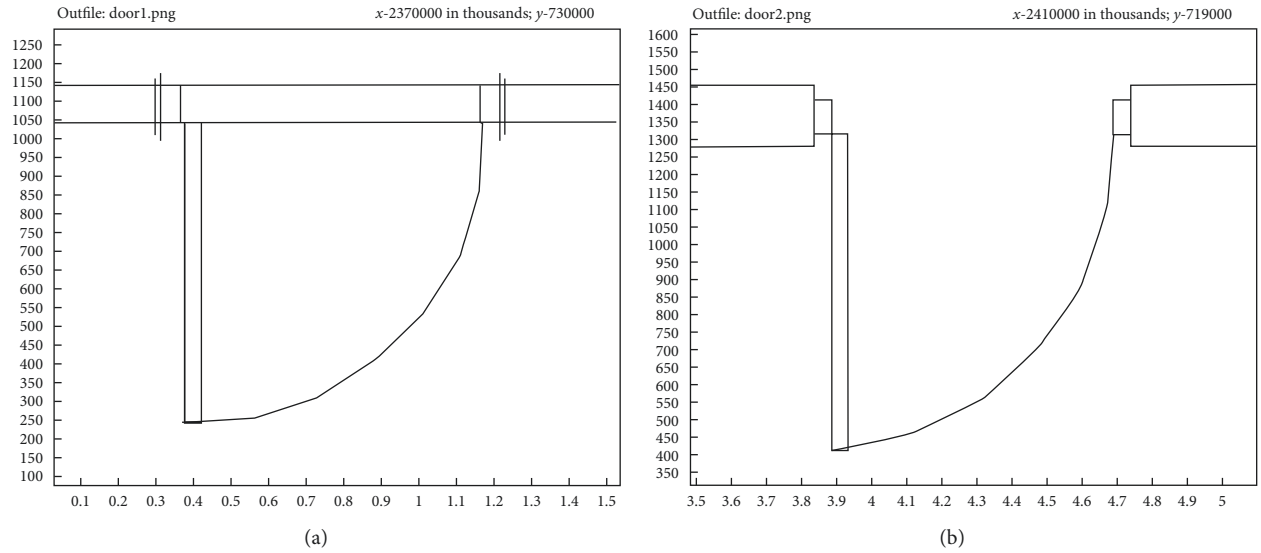



Figure 1: Typical door drawings that must be recognized by the heuristics. Approximating circular arcs by polygonal lines is artifact from the software used to create this figure. The horizontal scale is in meters, and the vertical scale is similar (but indicated in millimeters).

the data structure from [9] is well known, we need to only explain how to obtain a good set of wall and door segments.

We start by approximating ellipses, circular arcs, and splines by polygonal lines. Then we find all segment intersections and insert explicit intersection points using snap rounding as explained in [10]. There is also a need to be no overlapping segments, but having all intersections explicit makes it trivial to find them and remove duplicates.

It should be noted that buildings often allow one to walk from one room to another without encountering a swinging door. It is necessary to add door segments to the data structure in order to prevent large complicated regions from being erroneously treated as single rooms. Section 2.3 will explain how to do this. We perform this step immediately after replacing swinging doors by door segments.

We now have everything that is needed in order to construct a planar subdivision, but Figure 1 suggests a problem. Since walls are drawn with two parallel lines (and sometimes additional internal details), the planar subdivision will have a large number of faces that correspond interiors of walls. We simply need to postprocess the planar subdivision so that the remaining faces are actual rooms.

In order to distinguish room polygons from polygons that are just within walls, we use $2A/P$ as a generalized notion of average width, where $A$ is the polygon area and $P$ is the perimeter. We simply reject rooms for which $2A/P < D_0$, where $D_0$ is the minimum door size. To compute the minimum door size, we use the heuristics from Section 2.1 to find as many doors as possible, then compute the length of each door segment and find the 5th percentile.

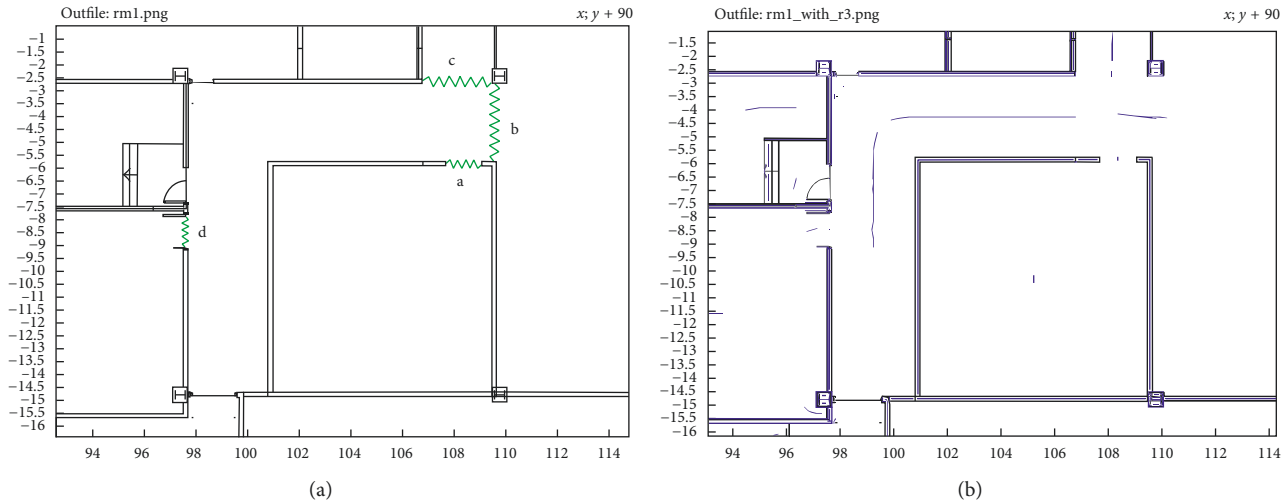
*2.3. Voronoi Diagrams for Doorway Features.* The door-finding heuristics from Section 2.1 eliminate most of the swinging door pictures and replace them with doorway segments. Figure 2(a) shows a portion of the result. The large room in the middle has a doorway but no swinging door, so no door segment was inserted. Jagged line $a$ shows where the segment should be, and jagged lines $b, c,$ and $d$ show where more door segments are desirable.

The places with missing doorways are quite similar to the "stroke-like features" from [11]. As in that paper, the process of finding such features is based on the Voronoi diagram for line segments. The intuitive idea is that such a diagram finds centers of hallways but generalizes the notion of hallway width so that missing doorways appear as places of constricted width.

Given a collection of line segments, we can divide the surrounding space into regions based on which the line segment is closest. The borders of such regions are composed of straight lines and portions of parabolas. Any point $P$ on the boundary between two of these Voronoi regions is equidistant from two input segments. Suppose $Q_L$ and $Q_R$ are the points where those segments most closely approach $P$. An important idea from [11] is that the *opposition angle* $Q_LPQ_R$ can be used to prune the Voronoi diagram. The portions of the Voronoi diagram where the opposition angle is close to 180° are the "generalized hallways," and the "generalized hallway width" is the Euclidean distance from $P$ to $Q_L$ or $Q_R$ (it does not matter which).

Throwing away portions of the Voronoi diagram where the opposition angle is <146° produces curvilinear paths that track the centers of hallways as well as the centers of walls and also the missing doorway features that we are looking for. This is how we generated the lines that were added to Figure 2(a) in order to generate Figure 2(b). As long as the threshold is >120°, it is impossible for three or more of the pruned Voronoi segments to meet. Hence the pruned Voronoi is composed of curvilinear paths for which the opposition angle is a function of distance along the path.

Figure 3 suggests how to use the pruned Voronoi paths to recognize openings where door segments are needed. When a Voronoi path goes through a doorway-type opening, the opposition angle increases and then decreases. We can




(a) (b)

Figure 2: (a) A portion of the building after using heuristics to replace swinging doors with door segments. The zigzag lines a, b, c, and d show where more door segments are needed. (b) The result of adding portions of the line segment Voronoi diagram where the opposition angle is <146°.

also look at the distance from any point on the Voronoi path to its $Q_L$ and $Q_R$ points. Define the *local hallway width* to be twice this distance. It decreases when going from $P_0$ to $P_1$, then it starts increasing and reaches 1.166 times its former value at $P_2$. This temporary decrease in the local corridor width is a good indication that a doorway segment is needed.

More precisely, the requirement for inserting a doorway at Voronoi path point $P_1$ is that the opposition angle there is >157° and

$$W(P_i) - W(P_1) > 0.24 \cdot \sqrt{4\left\|P_i - P_1\right\|^2 + W(P_1)^2} \quad \text{for } i = 0, 2, \tag{1}$$

where $W(\cdot)$ denotes the local corridor width and $P_0$ and $P_2$ are the points on the pruned Voronoi path where the temporary dip in the local corridor width begins and ends.

If the pruned Voronoi path is much shorter than the local corridor width (as it is at the entrance of the large room in Figure 2(b)), this is probably a defect in the Voronoi implementation. Algorithms for the line segment Voronoi diagram are famously difficult to implement reliably, so it is wise to be prepared for such defects. In [12], Held discusses these implementation problems and gives good techniques for minimizing them.

We avoided Held's implementation due to concerns about restrictions against commercial use. The alternative is beyond the scope of this paper—it is a more brute-force approach that asks for a many carefully chosen points $(x, y)$ "which segment's Voronoi region does $(x, y)$ belong to?" In particular, we cope with defects such as the short path in Figure 2(b) by performing "which segment is closest to $(x, y)$?" tests for points $(x, y)$ beyond the ends of the truncated path. This tells what the local corridor width would be at these $(x, y)$ points so that these points can be used as the $P_0$ and $P_2$ points in (1).

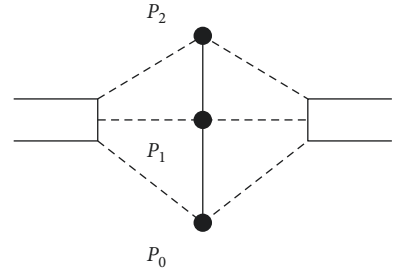



Figure 3: The typical behavior of the pruned line segment Voronoi for a missing doorway opening. The opposition angle increases from 120° to 180° as you go from $P_0$ to $P_1$, then it drops to 120° at $P_2$. The distances from the path to the doorway are 1.166 times as high at $P_0$ and $P_2$ as at $P_1$.

## 3. A Combined Tracking

Although some of the ideas discussed in this paper could apply to a variety of tracking technologies, we focus on RSSI measurements (radio signal strength) for various Wi-Fi access points. For various known locations in the building, a "fingerprint database" provides expected values for a signal-strength vector that gives RSSI versus MAC address for each access point. The dimensionality of the signal-strength vector may be fairly high, for example, 50.

The design goal for the tracking algorithm is to retain the advantages of Kalman filter tracking, while adding room-based analysis. Hence, there is a two-part implementation:

(i) The Kalman filter algorithm (implemented in C++) with minor modifications to facilitate multiple copies of the algorithm running on various alternative rooms

(ii) A supervisor that takes the particles from the Kalman filter and infers likelihoods for the questions "Is it in this room?" and "Is it near a door?"

Section 3.1 briefly outlines the fundamentals of the Kalman filter algorithm and explains how to use smoothed fingerprint measurement data. The Gaussian process smoothing is just one possible source of smoothed data. The approach is general enough to handle any source of RSSI measurements and uncertainty estimates.

Section 3.2 explains how a refined mesh based on Delaunay triangulations can reduce some of the noise in the RSSI values for fingerprint points, while estimating which data points have the most uncertainty. This avoids the $O(n^3)$ behavior of the Gaussian process smoothing and takes walls into account. A wall between two fingerprint database points makes it more reasonable for their RSSI values to differ.

Section 3.3 describes the modifications to make the Kalman filter tracking work well with multiple rooms. Some computations are done on a per-room basis, and particle weights are not normalized separately for each room.

Section 3.4 describes the room chooser and how it interfaces to the Kalman filter algorithm. For the purposes of the following high-level algorithm description, its primary output is a set of alternative rooms with probabilities for each:

(1) Refine the fingerprint data as explained in Section 3.2. This is Algorithm 1.

(2) Initialize the set of rooms to all possible rooms with equal probabilities for each.

(3) Perform a room-based Kalman filter step as explained in Section 3.3.

(4) Use Algorithm 2 from Section 3.4 to update the set of rooms and the probability of being in each room. If not all done, go back to Step (3).

*3.1. A Priori Probabilities for Kalman Filter Tracking.* The object being tracked has various possible trajectories, and we use a set of particles to represent the distribution of these trajectories. Each particle has a position, a velocity, and a weight that represents the relative likelihood for that trajectory. At each time step, we need a quick way to compute a probability for each particle based on how well the observed RSSI values fit the fingerprint data for the particle's position. Call this the *a priori particle probability*. The fundamental idea behind Kalman filter tracking is multiplying this probability into the particle weight and renormalizing the particle weights at each time step. Random Gaussian perturbations to the particle accelerations ensure that the particle cloud covers the entire distribution of possible trajectories.

Let the fingerprint database be

$$\mathscr{D} = \{(L_1, R_1), (L_2, R_2), \ldots, (L_n, R_n)\}, \tag{2}$$

where each $R_j$ is an RSSI vector and $L_j$ is the corresponding location in the building and the $L_j$'s are all different. We need to be able to use $\mathscr{D}$ to compute an a priori particle probability for any location $L$ and any measured RSSI vector $S = s_1, s_2, \ldots, s_k$. In other words, we need to select or interpolate from $\mathscr{D}$ an RSSI vector $R = r_1, r_2, \ldots, r_k$ that is appropriate for $L$, and we need a vector $E = e_1, e_2, \ldots, e_k$ of uncertainty estimates. Treating the components of $E$ as standard deviations gives

$$\sum_{i=1}^{k} \left( \frac{r_i - s_i}{e_i} \right)^2. \tag{3}$$

Then the appropriate way to convert a result $s$ for this sum of squared relative errors into a probability is to use $e^{-s/2}$.

A common way to extend such a fingerprint database $\mathscr{D}$ to more $(x, y)$ locations is via Gaussian process (GP) smoothing. Ferris et al. explained GP in the context of RSSI-based tracking [6]. For our purposes, the key feature of GP is that it can extend (2) to

$$\overline{\mathscr{D}} = \{(\overline{L}_1, \overline{R}_1, \overline{E}_1), (\overline{L}_2, \overline{R}_2, \overline{E}_2), \ldots, (\overline{L}_m, \overline{R}_m, \overline{E}_m)\}, \tag{4}$$

where the locations $\overline{L}_j$ can be chosen to facilitate tracking and each RSSI vector $\overline{R}_j$ is accompanied by a vector of uncertainties $\overline{E}_j$.

For instance, the $\overline{L}_j$ points may be chosen so that they define a regular triangular mesh: tile the building with equilateral triangles and use their vertices as the points $\overline{L}_j$ in (2). In order to evaluate (3) at a point $L$, just compute its barycentric coordinates in its equilateral. Suppose $L = \alpha\overline{L}_{j_1} + \beta\overline{L}_{j_2} + \gamma\overline{L}_{j_3}$, where $\alpha, \beta$, and $\gamma$ add up to 1 and are all nonnegative. Then we can use

$$\begin{aligned} R &= \alpha\overline{R}_{j_1} + \beta\overline{R}_{j_2} + \gamma\overline{R}_{j_3}, \\ E &= \alpha\overline{E}_{j_1} + \beta\overline{E}_{j_2} + \gamma\overline{E}_{j_3}, \end{aligned} \tag{5}$$

in (3).

It is worth noting that any distribution of locations $\overline{L}_j$ in (4) can be used to computing a priori particle probabilities. Each RSSI component (in dB) is a piecewise-linear function of $(x, y)$ if we proceed as follows:

(1) Triangulate the set of locations as well as possible, that is, compute the Delaunay triangulation

(2) Use a data structure that allows rapid identification of the containing triangle for any particle location $L$

(3) Use $L$'s barycentric coordinates in (5) to get $R$ and $E$ for (3)

Unlike the Voronoi diagram for line segments of Section 2.3, Delaunay triangulations of point sets are easy to compute reliably. We used Fortune's implementation of his own algorithm [13] for Step (1).

For Step (2), we used slab decomposition [14] because it is simple and allows fast $O(\log n)$ searches. Its $O(n^2)$ space preprocessing time costs can be avoided by more complicated methods, but we found this unnecessary.

*3.2. Smoother Fingerprint Data on a Refined Mesh.* A major drawback of the Gaussian process approach is that deriving (4) from (2) is very expensive in terms of processing time ($O(mn^3)$ if implemented in the most straightforward manner). Furthermore, it does not allow for any RSSI discontinuities due to interior walls even though radio propagation models generally treat such walls as partly reflective. See the study by Fortune et al. [15] for a discussion of indoor radio propagation.

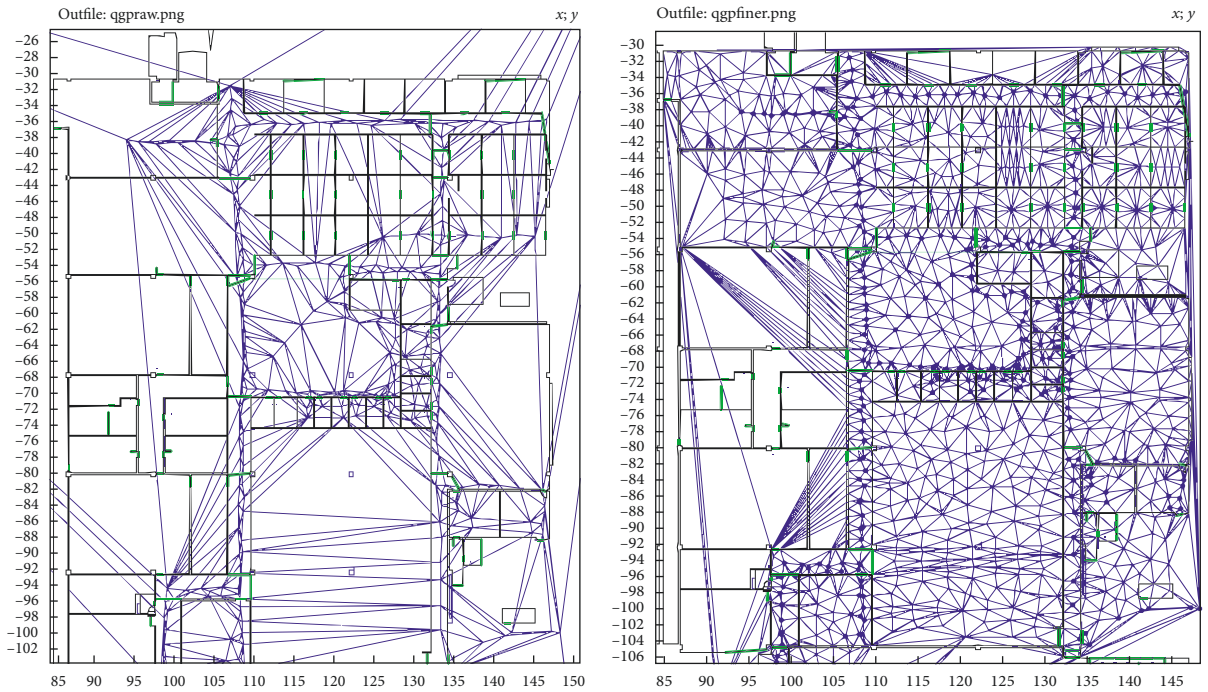



FIGURE 4: (a) The Delaunay triangulation of the unrefined fingerprint database locations with coordinates in meters; (b) a refined version, also Delaunay triangulated, with bold dots at the locations that are also (a) vertices.

As noted in Section 3.1, a priori particle probabilities can be computed directly from the fingerprint database $\mathscr{D}$ if we provide uncertainty vectors $\overline{E}_j$ (perhaps with all entries defaulted to some common value). Figure 4(a) hints at what is wrong with this approach: even with a Delaunay triangulation, many of the triangles are too big and elongated.

The piecewise-linear interpolation is too unsophisticated to be used directly on the large triangles of Figure 4 (a), so we need rules for refining them. These rules that contain heuristic parameters ($H$ with various subscripts) whose values will be discussed in Section 4 are as follows:

(1) Each Delaunay triangle's circumscribing circle must have a radius $\leq H_t$.

(2) No edge in the Delaunay triangulation may cross a wall at more than distance $H_e$ from a wall.

Rule 1 is a natural way to keep the triangles from getting too big because the Delaunay construction guarantees that the circumscribing circle is empty; that is, no vertex of any triangle is strictly within the circle. Rule 2 is needed in order to assign RSSI values to the refined mesh in a manner that takes walls into account. Figure 4(b) shows a refinement that is based on these rules.

Algorithm 1 achieves this refinement by enforcing the two rules. It starts with $n$ measurement point locations $L_j = \overline{L}_j$ that are retained in the refined fingerprint database. These $j \leq n$ have measured RSSI vectors $R_j$ that are ultimately replaced by adjusted vectors $\overline{R}_j$. New vertices $\overline{L}_j$ with $j > n$ are added until we reach some total $m > n$. The adjusted $\overline{R}_j$'s and the $\overline{R}_j$'s for the added vertices are obtained later by solving systems of linear equations in Step (7).

*Algorithm 1.* The fingerprint database refinement algorithm—use $\mathscr{D}$ to compute $\overline{\mathscr{D}}$.

(1) Initialize $m \leftarrow n$, $\overline{L}_i \leftarrow L_i$, and $\overline{R}_i \leftarrow R_i$ for $i \leq n$. $m$ will increase as more points are added. The relation $\overline{L}_i = L_i$ is invariant for all $i \leq n$, but $\overline{R}_1, \overline{R}_2, \ldots, \overline{R}_n$ will change. These changes correct $\overline{R}_1, R_2, \ldots, \overline{R}_n$ to be more consistent with their neighbors.

(2) Add $\overline{L}_j$'s for all the vertices of each room polygon. Sort them by $x$ coordinate, find pairs that are within $\infty$-norm distance $H_t$, and retain only one $\overline{L}_j$ for each such pair.

(3) Find the Delaunay triangulation of all the $\overline{L}_j$'s.

(4) Find the circumscribing circle for each Delaunay triangle and add a new $\overline{L}_j$ for the center of each circle whose radius exceeds $\leq H_t$.

(5) Add new $\overline{L}_j$'s where Delaunay edges cross walls as required by Rule 2.

(6) Go back to Step (3) if any $\overline{L}_j$'s were added in Steps (4) and (5).

(7) For each access point, solve a sparse linear system whose $m$ unknowns are that the access point's component in the $\overline{R}_1, \overline{R}_2, \ldots \overline{R}_m$ vectors. The system sets each $\overline{R}_j$ to a weighted average of $\overline{R}_{j'}$'s for its Delaunay neighbors with weights as explained near

Table 2: Room and doorway scores for one of the room transitions on the "Dublin 87pts forward" track.

| Room 39 | Door 39 → 35 | Door 39 → 70 | Door 39 → 19 | Room 70 |
|---|---|---|---|---|
| 0.9945 | $4 \times 10^{-141}$ | 0.06177 | 0.08559 | 0 |
| 1 | $4\times10^{-19}$ | $5 \times 10^{-09}$ | $9 \times 10^{-06}$ | $9 \times 10^{-09}$ |
| 0.9820 | $3 \times 10^{-12}$ | 0.00099 | 0.0080 | 0 |
| 0.9998 | $9 \times 10^{-45}$ | 0.3088 | 0.3564 | 0.00017 |
| 0.00134 | $2 \times 10^{-51}$ | 0.9666 | 0.518976 | 0.9987 |
| $6 \times 10^{-15}$ | $6 \times 10^{-50}$ | 0.7975 | 0.04788 | 1 |
| $2 \times 10^{-32}$ | 0 | 0 | 0 | 1 |

Each row is one time step (many seconds since the tester repeatedly stopped to gather data).

the end of Section 3.2. If $j \le n$, the $R_j$ counts as a neighbor of $\overline{R}_j$.

(8) For $j = 1, 2, \ldots, m$, set each component of $\overline{E}_j$ to $\sqrt{H_c^2 + (H_s\overline{d}(j))^2 + (H_v\widetilde{e}(j))^2}$ where $\overline{d}(j)$ is the distance from $L_j$ to the closest $L_1, L_2, \ldots, L_n$ point and $\widetilde{e}(j)$ is the $R_i - \overline{R}_i$ for that closest point.

The weights for averaging neighboring RSSI values in Step (7) are basically $1/d$ where $d$ is the distance from $\overline{L}_j$ to the neighboring vertex. If the neighbor is in a different room, the weight is $1/(d + H_p)$. If $j \le n$ and we are treating the $R_j$ measurements as a neighbor of the adjusted values $\overline{R}_j$, the weight is $1/H_m$.

For Step (8), the closest measurement point to a vertex location $\overline{L}_j$ is found via a breadth-first search that adds $H_p$ to the length of any Delaunay edge that crosses from one room to another. If it is convenient to find (say) the closest 4 measurement points $L_{i_1}, \ldots, L_{i+4}$, then $\widetilde{e}(j)$ might be modified to return a weighted average of $R_{i_1} - \overline{R}_{i_1}$ through $R_{i_4} - \overline{R}_{i_4}$ with the weights based on inverse distance.

*3.3. Kalman Filter Tracking for Multiple Rooms.* Recall that Kalman filter tracking multiplies each particle weight by its a priori particle probability (as explained in Section 3.1), and the total weight is renormalized at each time step.

It seems safe to assume that if we know what room to look in, the tracking problem should be easier. This leads to a multiple-room version of the Kalman filter tracking where each particle is constrained to a specific room and the renormalization of total particle weight at each time step is done once for all rooms, not separately for each room being considered. Hence, the total particle weight in a given room is a likelihood estimate for whether the tracked object is in the room:

(i) Each particle has a location $P_i$, a velocity vector $V_i$, a weight $w_i$, and a room number $r_i$.

(ii) The effective particle count $(\sum w_i)^2/\sum w_i^2$ determines whether particles need to be regenerated, but this decision is made separately for each room. The room's total particle weight is preserved.

(iii) There is a linked list of particles for each active room, and they are freed when the room chooser deselects a room.

*3.4. Using Room and Doorway Scores to Supervise Multiroom Tracking.* The input to the room chooser is a set of particles: the results from Kalman filter tracking at the last time step. The output is a list of room numbers for the next time step, with a suggested location and total particle weight for each selected room. The locations and weights are to be used if the Kalman filter tracker has no suitable left-over information about the room.

The room chooser also needs private data structures to record recent results that are needed for future decisions. For some fixed maximum number of prior time steps such as Nhist = 16, each room has scores ts[ ] and a similar array of *doorway scores* for each door that estimates, for each time step, the conditional probability of being near the door if in the room. Each room also has a "centroid location" for each time step in case the Kalman filter needs to reinitialize for that room and has no existing particles on which to base a position.

The algorithm that uses these data structures for room choosing at one time step involves linear regression. The doorway scores depend on per-room positions that are obtained by averaging particle $P_i$ positions separately for each room. Each doorway score is computed as follows:

(1) Apply linear regression to the distance from the doorway, obtaining a distance estimate for the current time step

(2) Compute a position uncertainty based on the room's variance of $P_i$ positions with added terms proportional to the room dimensions

(3) Use $e^{-\rho^2}$ where $\rho$ is the ratio of distance to uncertainty, but multiply by a term that considers the speed of approach toward the door

Table 2 shows how some of the doorway scores vary as a function of the time step along the "Dublin 87pts forward" test track. All the scores are functions of time, and it is natural to smooth them by applying linear (or logistic) regression. This is trivial because there is a single independent variable. For logistic regression, we transform the probabilities via $y = \ln(p/(1-p))$ then use $p = e^y/(1+e^y)$ to convert the resulting $y$ into a probability. The room choosing algorithm (Algorithm 2) uses (logistic) regression and an array $z[\ ]$ of room weights.

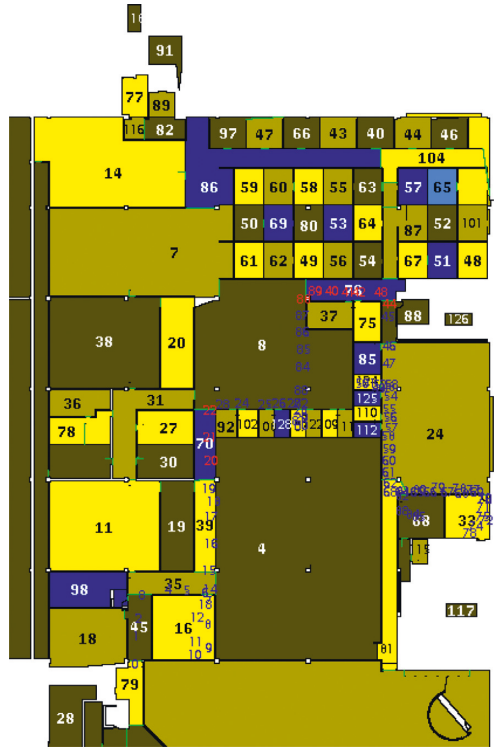



Figure 5: The relevant half of the Dublin building with room polygons shaded to provide a contrast between adjacent rooms. Smaller numbers $1, \ldots, 87$ show the "Dublin 87pts forward" track. ("Dublin 85pts reverse" covers almost the same locations, but in the reverse order.)

*Algorithm 2.* The room choosing algorithm.

(1) Compute doorway scores as explained above and use regression to compute a weight $z[r]$ for each room $r$.

(2) For each doorway, use regression to compute a doorway score $s_{ro}$ for each doorway from a room "$r$" to another room "$o$."

(3) For each doorway score, assign $z[o] = z[o] + s_{ro}z[r] \times (1 - z[o])$. This multiplies $1 - z[o]$ by $1 - s_{ro}z[r]$.

(4) Let $\overline{z}$ be the sum of all room weights $z[r]$.

(5) Sort rooms by descending $z[r]$ and select up to 10 of the high-weight rooms, stopping early if necessary to exclude rooms with $z[r] < \overline{z}/10^4$. (Of course, 10 and $10^4$ could be replaced by other constants.)

## 4. Results

All tests were based on a fingerprint database of 317 locations in the Dublin building as shown in Figure 4(a). Measurements at each location included RSSI values for 51 access points, some of which had multiple MAC addresses. Measurements at each location were repeated 5 times at approximately 1 second intervals so that each RSSI value in the database is the average of at least 5 measurements.

Figure 5 shows how the Dublin building is divided into room polygons There were two test tracks: "Dublin 87pts forward" passes through rooms 45, 35, 16, 35, 39, 70, 8, 107, 8, 76, 88, 24, 124, 81, 24, 33, 24, and 68 and "Dublin 85pts reverse" passes through rooms 81, 68, 24, 112, 24, 85, 88, 76, 87, 104, 86, 14, 7, 8, 108, 8, 111, 8, and 37. The test tracks also had 5 measurements per position; that is, the test subject stopped 87 times to take measurements and recorded the ground truth location. The building actually has a fairly open floor plan, so many of the room boundaries in Figure 5 are partial partitions or places where hallways have 90° bends; for example, rooms 45, 35, 39, and 70 in Figure 5 are all one hallway.

The ground truth locations for the test tracks came close to convex hull of the fingerprint database locations, so it was essential to have a reasonable idea of what the RSSI values should do beyond the convex hull. Simply extrapolating based on Figure 4(b)'s outermost Delaunay triangles is not adequate. We solved this problem by augmenting the fingerprint database with four artificial data points, one at just outside each corner of the building. The artificial points have a common RSSI value $H_q$ for each Wi-Fi access point being measured.

We used "Dublin 87pts forward" track to optimize $H_q$ and the $H_c$, $H_m$, $H_s$, $H_v$, $H_p$, $H_t$, and $H_e$ parameters from Section 3.2. Since the $H_q$ parameter belongs to the tracking program which is considerably faster than the fingerprint refinement algorithm of Section 3.2, we tried 11 values of $H_q$ for each run of the refinement algorithm. We give this 7-dimensional optimization problem to a Nelder-Mead optimizer, and after 588 evaluations, it selected the values in Table 3. The table includes $H_q$ which was hidden from Nelder-Mead and tested exhaustively as explained above (see [15, 16] for a discussion of Nelder-Mead optimization).

Table 3: Heuristic parameter values that optimize "Dublin 87pts forward" results.

| Parameter | Value | Description |
|---|---|---|
| $H_c$ | 10.66 | Constant term for RSSI variance |
| $H_m$ | 0.194 | Distance at which mesh edge is as strong as tie to measurements |
| $H_s$ | 0.166 | RSSI variance term per meter separation from real data point |
| $H_v$ | 1.050 | Weight for neighbor variances in RSSI variance |
| $H_p$ | 10.40 | Distance penalty for edges that cross room boundaries |
| $H_t$ | 2.107 | Refined Delaunay triangles must fit in circle of this radius |
| $H_e$ | 0.411 | Don't allow Delaunay edge across a wall < this from a vertex |
| $H_q$ | −95 | Fingerprint RSSI at artificial corner points |

All distances are in meters.

Table 4: RMS error in meters and room accuracy for various test runs (including the weaker condition that any of the algorithm's top 3 rooms is the correct one).

| Test track | $H_q$ | RMS error (meters) | Top room OK | Among top 3 |
|---|---|---|---|---|
| Dublin 87pts forward | −95 dBm | 3.43 | 64.4% | 95.4% |
| Dublin 85pts reverse | −98 dBm | 6.82 | 54.1% | 76.5% |
| Dublin 36pts reverse | −93 dBm | 3.66 | 55.6% | 91.7% |
| Dublin 85pts reverse | −95 dBm | 12.03 | 29.4% | 51.8% |
| Dublin 36pts reverse | −95 dBm | 12.55 | 44.4% | 55.6% |

One more heuristic parameter $H_f$ was set based on early test runs. This relates to the formula $e^{-s/2}$ for converting the sum $s$ from (3) into an a priori particle probability. To make the probabilities less wild, the actual formula was $e^{-sH_f/2}$, where $H_f = 0.45$.

Table 4 shows tracking accuracy for the test tracks. Results depend strongly on $H_q$, probably because the artificial corner points do not give a very good idea of what RSSI values to expect outside the bounding box of the fingerprint database locations. The "Dublin 36pts reverse" track is the last part of the "Dublin 85pts reverse" track. It was introduced to show that performance can be close to the 3.43 meter optimized value when similar heuristic parameters are used for a different test track.

## 5. Conclusions

A big part of room-based tracking is simply finding the room polygons and constructing a planar subdivision along with the doorways that tell how it is possible to move from room to room. We have seen how the existing approach of extracting walls and doors from CAD drawings can be extended to construct planar subdivisions. The line segment Voronoi diagram is a critical tool for recognizing places where additional doors have to be added. Even if the CAD drawing yields perfect information about doors, the additional doors are needed in order to prevent room polygons from excessively getting large and complicated.

This specific strategy of running Kalman filters in separate rooms is just one of many possible ways to use a room-based planar subdivision. The scheme needs more testing and various engineering improvements to make it more reliable. The reason for retaining multiple guesses as to which room is current is to avoid situations where a wrong choice of starting room cascades to later time steps due to doorway constraints that do not allow travel to the correct room. The results in Section 4 suggest a need for more ways to recover from a bad initial room. Perhaps there should be a restart mechanism.

Another problem that is apparent from Section 4 is that no one value of the corner point RSSI parameter $H_q$ works well in all cases. It is a nontrivial problem to cope with cases where the test track nearly exits the convex hull of the fingerprint measurement locations. The RSSI values at the artificial corner points could be set more intelligently by looking at the fingerprint data near the edge of the convex hull and making separate decisions about each access point. Another option would be to compute the distance to the convex hull and decrease fingerprint RSSI values by some appropriate function of distance. Perhaps the best approach would be to have Section 3.2 solve the problem by extending the refined mesh further outward and adjusting the linear system in Step (7) of Algorithm 1 to handle the extrapolation in some intelligent manner.

Ideally, the measurement points in the fingerprint database should extend far enough beyond the test track so that we do not need a priori particle probabilities beyond the convex hull of the measurement points. Of course it would help to have more fingerprint measurement points, but there are strong incentives to reduce the work required to gather and update such data. Indeed, the purpose of Section 3.2 (or Gaussian process smoothing) is to cope with sparse fingerprint data.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## References


[1] M. Dashti and H. Claussen, "Extracting location information from rf fingerprints," in *Proceedings of 2016 IEEE Globecom Workshops (GC Workshops)*, IEEE, Washington, DC, USA, December 2016.

[2] R. Piché and M. Koivisto, "A method to enforce map constraints in a particle filter's position estimate," in *Proceedings of Positioning, Navigation and Communication (WPNC), 2014 11th Workshop*, pp. 1–4, IEEE, Dresden, Germany, March 2014.

[3] M. Schäfer, C. Knapp, and S. Chakraborty, "Automatic generation of topological indoor maps for real-time map-based localization and tracking," in *Proceedings of Indoor Positioning and Indoor Navigation (IPIN), 2011 International Conference*, pp. 1–8, IEEE, Guimaraes, Portugal, September 2011.

[4] M. Klepal, S. Beauregard, and Widyawan, "A novel backtracking particle filter for pattern matching indoor localization," in *Proceedings of the First ACM International Workshop on Mobile Entity Localization and Tracking in GPS-Less Environments*, pp. 79–84, ACM, San Francisco, CA, USA, December 2008.

[5] M. Kessel and M. Werner, "Automated WLAN calibration with a backtracking particle filter," in *Proceedings of International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1–10, Alcala de Henares, Spain, October 2012.

[6] B. Ferris, D. Haehnel, and D. Fox, "Gaussian processes for signal strength-based location estimation," in *Proceedings of Robotics Science and Systems*, Citeseer, Cambridge, MA, USA, June 2006.

[7] H. Nurminen, A. Ristimaki, S. Ali-Loytty, and R. Piché, "Particle filter and smoother for indoor localization," in *Proceedings of International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1–10, Alcala de Henares, Spain, October 2013.

[8] L. Wirola, T. A. Laine, and J. Syrjärinne, "Mass-market requirements for indoor positioning and indoor navigation," in *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference*, pp. 1–7, IEEE, Zürich, Switzerland, 2010.

[9] L. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams," *ACM Transactions on Graphics*, vol. 4, no. 7, pp. 74–123, 1985.

[10] J. D. Hobby, "Practical segment intersection with finite precision output," *Computation Geometry Theory and Applications*, vol. 13, no. 4, pp. 199–214, 1999.

[11] J. D. Hobby, "Generating automatically tuned bitmaps from outlines," *Journal of the ACM*, vol. 40, no. 1, pp. 48–94, 1993.

[12] M. Held, "VRONI: an engineeering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments," *Computational Geometry: Theory and Applications*, vol. 18, no. 2, pp. 95–123, 2001.

[13] S. Fortune, "Sweepline algorithms for Voronoi diagrams," *Algorithmica*, vol. 2, no. 1–4, pp. 153–174, 1987.

[14] D. Dobkin and R. J. Lipton, "Multidimensional searching problems," *SIAM Journal on Computing*, vol. 5, no. 2, pp. 181–186, 1976.

[15] S. J. Fortune, D. M. Gay, B. W. Kernighan, O. Landron, R. A. Valenzuela, and M. H. Wright, "Wise design of indoor wireless systems: practical computation and optimization," *IEEE Computational Science and Engineering*, vol. 2, no. 1, pp. 58–68, 1995.

[16] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.