Algorithmica (2000) 26: 515–559
DOI: 10.1007/s004539910023

Algorithmica

Mobile Robot Self-Localization without Explicit Landmarks[1]

R. G. Brown[2] and B. R. Donald[3]

Abstract. *Localization* is the process of determining the robot's location within its environment. More precisely, it is a procedure which takes as input a geometric map, a current estimate of the robot's pose, and sensor readings, and produces as output an improved estimate of the robot's current pose (position and orientation). We describe a combinatorially precise algorithm which performs mobile robot localization using a geometric model of the world and a point-and-shoot ranging device. We also describe a rasterized version of this algorithm which we have implemented on a real mobile robot equipped with a laser rangefinder we designed. Both versions of the algorithm allow for uncertainty in the data returned by the range sensor. We also present experimental results for the rasterized algorithm, obtained using our mobile robots at Cornell.

Key Words. Navigation, Mobile robots, Rasterized algorithms, Localization.

1. Introduction.

Localization is the process of determining the robot's location within the environment. More precisely, it is a procedure that takes as input (i) a map, (ii) an estimate of the robot's current pose, and (iii) a set of sensor readings and produces as output a new estimate of the robot's current pose. Any of (i), (ii), and (iii) may be incomplete, and all are toleranced by error bounds. For example, the map may lack information, the pose estimate may contain only orientation information, and the set of sensor readings may have elements with impossible values. The output estimate can be incomplete, disconnected, or toleranced. By *pose*, we mean either (a) the position and orientation of the robot in the world or (b) the translation and rotation necessary to make a set of sensor readings (or a local map, built from those readings) best match an a priori global map.

In this paper we present algorithms which for estimating a mobile robot's pose by computing sets of poses which provide a maximal-quality match between a set of current sensor data and a map. For ease of exposition, and in the interest of completeness, we

[1] This paper describes research done in the Robotics and Vision Laboratory at Cornell University. Support for our robotics research is provided in part by the National Science Foundation under Grant Nos. IRI-8802390, IRI-9000532, IRI-9201699, IRI-9530785, CDA-9726389, 9802068, CISE/CDA-9805548, EIA-9818299, IRI-9896020, EIA-9901407, IIS-9906790, and by a Presidential Young Investigator award to Bruce Donald, in part by the Air Force Office of Sponsored Research, the Mathematical Sciences Institute, Intel Corporation, AT&T Bell laboratories, an equipment grant from Microsoft Research, and in part by a grant from Sandia National Laboratories as part of SNL Contract No. AJ-7129.
[2] TurnStyle Segmented Wooden Turnings, 7716 El Conde Ave. NE, Albuquerque, NM 87110, USA. rgbrown@swcp.com.
[3] Department of Computer Science, Dartmouth College, 6211 Sudikoff Laboratory, Hanover, NH 03755-3510, USA. brd@cs.dartmouth.edu.

Received November 15, 1996; revised January 13, 1998. Communicated by R. Motwani and P. Raghavan.

first treat the localization problem as a precise combinatorial problem in computational geometry. We then show how the problem can be solved using rasterized algorithms. The rasterized algorithms, which are drawn from the same high-level concepts as the combinatorial algorithms, can be used by real mobile robots with limited computational and sensing resources. Finally, we present experimental results for the rasterized algorithm, obtained using our mobile robot, LILY, at the Cornell Computer Science Robotics and Vision Laboratory. This paper is based on a portion of the Cornell Computer Science doctoral thesis [Bro].

The main concept underlying the localization algorithms presented in this paper is that of the *feasible pose*. A feasible pose is one that is consistent with the available range and map information: our algorithms partition the robot's configuration space into poses that conflict with available range information and poses that do not; these latter are the feasible poses. Poses are feasible or infeasible relative to several parameters: a pose is feasible with respect to a map and a range vector if that pose places the robot such that the range vector terminates at an obstacle boundary and is otherwise unobstructed. A pose is feasible with respect to a map and a *set* of range vectors if it is feasible with the map and each of those individual range vectors. We can determine feasibility within an error bound: to do this, we allow the length of a range vector to vary within an uncertainty bound and determine the poses that are consistent with some pose plus-or-minus the uncertainty bound. We can extend feasibility within an error bound for multiple range readings, as well. We can also determine the minimum error bound for which there exists a feasible pose. Finally, with multiple range readings, we can count how many range readings are feasible with a given pose and map, and look for poses that are feasible with a maximal number of the supplied range readings.

How do we compute the feasible poses for a particular map M and range probe z? At an intuitive level, the procedure is to find all locations for the robot where its rangefinder would return z in the world M represents. Treat z as a vector whose tail is located at the robot and whose head lies at an obstacle boundary: if z can be situated with its tail at a point p, such that the head of z touches an obstacle boundary, but no obstacle boundary intersects z anywhere *other* than its head, then p is a feasible pose for map M and range probe z. Figure 1 shows how our method of localization works. We denote the feasible poses for a map M and a single range probe, z (resp. a set of range probes, Z) by $FP(M, z)$ (resp. $FP(M, Z)$). Part (a) of Figure 1 shows a map, M. Parts (b), (d), and (g) show three vectors, x, y, and z, that represent three different range probes. Part (c) shows $FP(M, x)$: the solid black lines are $FP(M, x)$, because those are all of the positions in M where we can place the tail of x so that its head touches an obstacle, but its body does not. Parts (e) and (h) show $FP(M, y)$ and $FP(M, z)$. Part (f) shows $FP(M, \{x, y\})$: the two black dots are the locations in M where $FP(M, x)$ and $FP(M, y)$ intersect. Since there are two dots, the two range probes, x and y, are insufficient to localize the robot in M uniquely. By taking a third range probe, z, though, the robot can be uniquely localized: part (i) shows the intersection of $FP(M, x)$, $FP(M, y)$, and $FP(M, z)$. This intersection is located at the single black dot. Therefore, $FP(M, \{x, y, z\})$ is that single point, so the robot must be located at that point.

In Section 2 we discuss the feasible pose concept at length and provide algorithms to compute feasible poses. We give an exact combinatorial algorithm to compute feasible poses for a set of range probes and a map specified as a set of polygons, for the

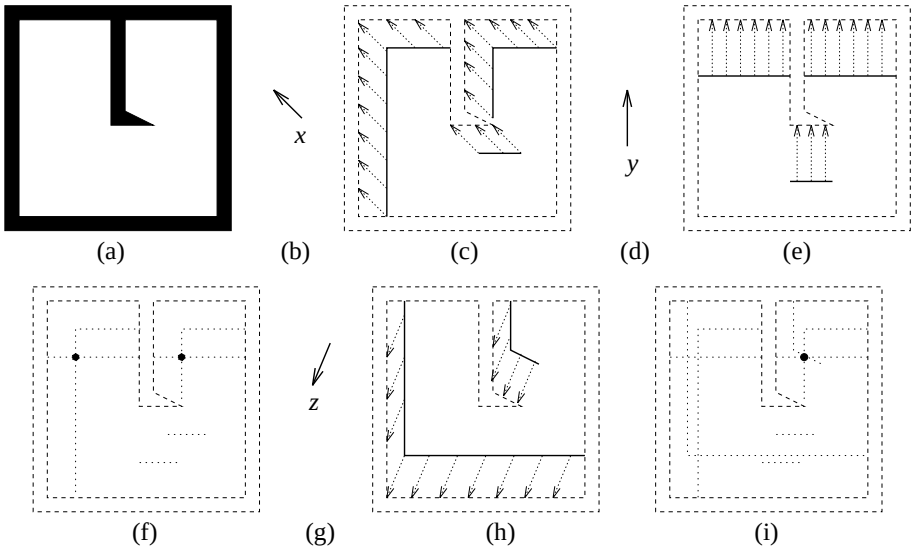


Fig. 1. How the localization algorithm works: Given (a) a map M and (b) a range vector x, we can determine (c) the set of feasible poses $FP(M, x)$. (d) and (e) show y and $FP(M, y)$. The black dots in (f) are $FP(M, \{x, y\})$. (g) and (h) show z and $FP(M, z)$. (i) shows the single point which is $FP(M, \{x, y, z\})$. The robot must occupy that position to get range vectors x, y, and z in map M.

case where the robot has translational uncertainty (unknown x–y position) but no rotational uncertainty (the robot knows which way it is facing). This algorithm makes use of tools from computational geometry. We provide analysis of the algorithm, and describe how it can be extended to the case where the robot *does not* know which way it is facing. We then present and analyze algorithms for computing feasible poses in the *rasterized* domain; in this case, the map is a binary array where ones represent obstacle cells and zeros represent empty cells. We feel that rasterized algorithms are more appropriate for solving many robotics problems; one reason for this opinion is that, in practice, rasterized algorithms often are easier to implement and run faster than their exact counterparts. Figures 2 and 3 show maps of our laboratory (a hand-drawn map for reference, and a map generated by our mobile robot LILY) and the output of the rasterized localization procedure for the map in Figure 2(b) and instantaneous range data. The robot-generated map is a statistical occupancy grid, defined in the next paragraph. The map was generated by the robot moving to a series of separate locations, sweeping a point-and-shoot laser rangefinder around in a circle while taking range readings, and performing updates to the occupancy grid based on those readings. For more examples, see Section 3. Table 1 summarizes the asymptotic complexities of the various localization algorithms. Note that all of our algorithms handle uncertainly in sensor measurements robustly.

The Underlying Spatial Representation. The map-making technique we chose for this system is a variant of the *statistical occupancy grid*. Moravec and Elfes introduced this method of map-making in [Elf], [Mor], and [ME] as a way of making detailed geometric maps using noisy sensors (sonar rangefinders, in their case). A simple occupancy grid

Fig. 2. The circles in these figures show the actual location and size of the robot. The circles are overlaid on (a) the hand-drawn map and (b) LILY's map.

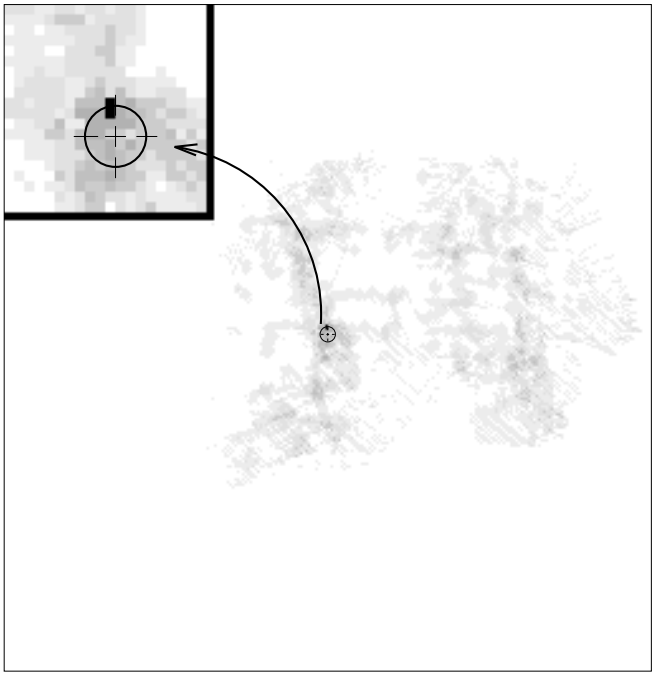


Fig. 3. The output of the localization procedure. In this picture the darkness of each cell is proportional to the number of range readings for which that cell is feasible. The black cells are the most feasible: these are the most likely positions for the robot. The circle overlaid on the output shows the actual location of the robot for this localization test.

Table 1. The Time-complexity of our algorithms.*

	One probe	m Probes
Exact combinatorial	$O((n+s)\log n)$	$O((mn^2+s)\log(mn))$
Rasterized	$O(n^2)$	$O(mn^2)$

*For the combinatorial algorithms, n is the number of features in the map, m is the number of range probes, and s is the number of intersections between features, which is $O(m^2n^2)$ worst case. For the rasterized algorithms, the map has size $n \times n$.

is a bitmap, where ones represent occupied cells and zeroes vacant cells. The main idea behind the statistical occupancy grid is that, since range data is inexact, the contents of each cell should be a probability: a cell containing a high probability is likely to be occupied, while a cell with low probability is likely empty. Moravec and Elfes use an update rule based on Bayesian probability to maintain the contents of each cell based on range readings that impinge upon that cell. We chose to adapt their work for a number of reasons, but primarily because a grid-based map-making technique fit best with our ideas about rasterized computational geometry for robotics (as discussed in Section 2). Figure 2(b) shows an occupancy-grid map of our lab generated by LILY using range data obtained from her laser rangefinder.

Experimental Work. In Section 3 we will discuss the results of some experiments we have performed with our mobots. We chose these experiments to allow us to assess the performance of our algorithms. We show maps of a small office and part of a large laboratory made using our laser rangefinder to build statistical occupancy grids. We also show results of running the localization algorithms described in Section 2 from locations within these environments, using laser rangefinder data and those maps.

1.1. *Related Work in Navigation, Localization, and Map-making.* A number of papers have been written in the last 8 years or so about localization (or pose estimation). Most authors, including ourselves, appear to be using roughly the same informal definition of pose estimation, but there is considerable difference in the sensor and world models under which the various authors perform their pose estimation.

At the theory level, the primary work to which we wish to compare our localization algorithms is that of Guibas et al. [GMR]. They present the localization problem at a theoretical level: The input is a world polygon and a visibility polygon. The visibility polygon is a star-shaped polygon representing the output of a swept rangefinder. They describe an algorithm which preprocesses the world polygon so that a localization query (in the form of a visibility polygon) can be answered in optimal time, $O(m + \log n + A)$, where m and n are the number of vertices in the visibility polygon and the world polygon, respectively, and A is the size of the output. The output, in this case, is a list of points at which the visibility polygon can be made to match the world polygon (i.e., feasible poses). Preprocessing the world polygon takes time $O(n^5 \log n)$ (worst case—expected case is apparently more like $O(n^2 \log n)$). A single-shot query (with no preprocessing) can be answered in time $O(mn)$. The main lacunæof this result are (i) there is no attempt to deal with uncertainty, (ii) the preprocessing assumes a static world. If the world model

changes at all, the preprocessing price must be paid again, and (iii) the model of a swept rangefinder as producing a visibility polygon is difficult to realize on a real robot. Note that these difficulties are not insurmountable, since the algorithm may well be adapted to compensate for these. A more fundamental difficulty with this algorithm is that, in general, obtaining the visibility polygon, as a list of vertices and edges, is extremely difficult with any physically realizable sensor.

In [Kle2] Kleinberg also presents an algorithm which approaches mobile robot localization from a theoretical viewpoint. He provides an on-line search algorithm by which a mobile robot moving within its environment can uniquely determine its location without access to an a priori map.

Many authors have described localization algorithms that make use of *beacons* or *landmarks*. These are typically features of the environment which it is easy for the robot to recognize. *Active beacons* are objects in the environment that emit a signal which the robot can sense; *passive beacons* are natural features in the environment, such as corners, straight line segments, and vertical edges, that the robot has a good chance of identifying. In [MR] a localization system consisting of a directional infrared detector system and a set of beacons that emit modulated infrared signals is described. If the robot can detect the angles to three beacons at known locations, it can use trigonometry to solve for its own position. Kleeman [Kle1] achieves a similar result using active sonar beacons—elapsed times between the receptions of chirps from the series of known-location emitters enable the robot to compute its location; in fact, he uses an Iterated Extended Kalman Filter [Jaz] to merge pose estimates based on the active beacons with that based on dead reckoning. Leonard et al. [LDWC] define a *geometric beacon* as "a special class of target which can be reliably observed in successive sensor measurements (a beacon), and which can be accurately described in terms of a concise geometric parameterization." The beacons they use are typically walls, corners, and cylinders. They use a Kalman Filter to combine the "measured" location of nearby beacons with the "expected" location (based on odometry and the robot's map) to compute new pose estimates. An example of outdoor beacon-based navigation is given in [KK]. Their beacons are such features as "peaks," "pits," "ridges," and "ravines"; these are appropriate to navigation in rough natural terrains.

Several attempts have been made to use computer vision to detect and locate beacons [CC], [AH], [RWA$^+$]. Chenavier and Crowley [CC] use an Extended Kalman Filter to estimate position from odometry, sonar data, and the location of visually detected landmarks. They select their landmarks by hand, choosing objects with strong vertical edges. This work is similar to that in [MR] and [Kle1], except that the landmarks here are extracted from an image. They support their work with rudimentary experimental evidence. Atiya and Hager [AH] present an algorithm that determines both the correspondence between observed landmarks (vertical edges in the environment) and an a priori map, and the location of the robot from that correspondence. They detect and locate their visual landmarks using stereo vision techniques. They present results of extensive experimentation in an uncluttered (empty) interior environment, indicating that their system works very reliably in this context. Roth et al. [RWA$^+$] use images and odometry in a very novel way: They compute *controlled hallucinations*, which are mockup three-dimensional renderings of their robot's immediate environment, based on an a priori map and a dead-reckoning estimate of pose. They then use a matching

algorithm to match real image edges to model edges, and update the pose estimate based on the model-to-image transformation.

Drumheller [Dru] did some of the early work in sonar-based localization. He obtains a large number of sonar readings, each with the transducer rotated a few degrees from the previous, then takes the polygon formed by the endpoints of these sonar "rays," and extracts straight line segments from it. He then uses the *interpretation-tree*-based two-dimensional matching algorithm of Grimson and Lozano-Pérez [GLP] and an a priori line-drawn map of the environment to produce *interpretations* (lists of possible (sonar segment/wall) pairings). Each interpretation corresponds to a pose for the robot. He then uses the *sonar barrier test*, which he defines as an additional constraint on sonar: that "an admissible robot configuration must not imply that any sonar ray penetrates a solid object." His algorithm returns the interpretation that passes the sonar barrier test and has the greatest amount of sonar contour in contact with the walls. Gonzalez et al. [GSO] use a similar approach, except that they use maps made by the robot (both a map made of line segments and a map of cells), use a laser rangefinder to obtain their sensory data, and use an iterative algorithm to match their maps against their sensory data. Holenstein et al. [HMB] use sonar data to perform localization: They extract "edge," "wall," "corner," and "unknown" features from this data, then find the transformation that maps each sensed feature onto a map feature of the same type. They then plot the transformations and look for clusters of similar transforms. This is similar to several matching algorithms based on the Hough Transform. See, for example, [SHD].

Beveridge and Riseman [BR] use a three dimensional, perspective-based computer vision matching algorithm to track a robot's progress as it navigates in hallways. Because of the physical and visual unclutteredness of hallway scenes, they are able to use the motion of detected edges to detect the robot's position and motion relative to the walls, corners, and doorways in its environment.

A sonar sensor which is capable of tracking environment features is presented in [MDW]. This sensor uses a noncollocated transmitter–receiver pair to track *regions of constant depth*. This sensor is used as part of a localization algorithm for a moving mobot in an indoor environment.

Moravec and Elfes introduced the *statistical occupancy grid* method of map-making in [Elf], [Mor], and [ME] as a way of making detailed geometric maps using noisy sensors (sonar rangefinders, in their case). A statistical occupancy grid is an array, where the contents of each cell is based on the confidence that the part of the environment corresponding to that cell is occupied. The main idea behind the statistical occupancy grid is that, since range data is inexact, the contents of each cell should be a probability: a cell containing a high probability is likely to be occupied, while a cell with low probability is likely empty. Moravec and Elfes use an update rule based on Bayesian probability to maintain the contents of each cell based on range readings that impinge upon that cell.

Much research has been done on motion planning and execution for mobile robots. Takeda and Latombe [TL], for example, approach the problem of motion planning with uncertainty by computing a *sensory uncertainty field* for each configuration q in the robot's configuration space. The sensory uncertainty field estimates the distribution of

possible errors in the robot's pose estimate when it is in configuration q. This information is used by a path planner to generate paths that minimize expected errors; the paths so generated are easier to execute reliably. Ratering and Gini [RG] describe a *hybrid potential field*, which combines a global, a priori potential field, generated from the robot's map with a local potential field generated from instantaneous sensory data. This hybrid potential field is designed to enable a mobile robot to execute a motion plan in the presence of unknown, moving obstacles.

2. The Mobile Robot Localization Algorithm.

2.1. *Introduction*. In this section we will describe in detail our approach to localization and also describe a family of algorithms for localization by mobile robots, beginning with precise computational geometric algorithms and progressing to algorithms which can be made to run efficiently on real mobile robots.

Before diving into formal, mathematical descriptions of localization algorithms, we define and "type" some of the terms and objects we will be using in this section. When we refer to a *map* in this section, we mean an a priori description of the environment, which will typically be either a collection of polygons supplied as lists of edges, or a bitmap (a two-dimensional binary array). A *range probe* or *range vector* is a vector whose length is the value returned by a rangefinder (typically, a laser rangefinder or other point-and-shoot distance measuring sensor) and whose orientation is the rangefinder's heading relative to the robot's coordinate frame.

Why do we need a localization algorithm? In the domain of robotic manipulator arms, position sensing is a sensing primitive—it is typically possible to get a position reading that is within some (small) error bound of the actual position of the manipulator. Furthermore, the error bound is a constant; it does not grow over time and applies over all configurations of the arm. With mobile robots, this assumption does not hold; odometry from wheel shaft encoders is fraught with error. One method for tracking the position of a moving object is *dead reckoning*: starting from a known position and computing current position by integrating velocity over time. This works poorly in practice: control and sensing uncertainty integrated over time make the dead reckoning estimate of position extremely inaccurate. Some work has been done toward developing localization techniques and algorithms that operate by comparing geometric representations of the robot's environment with instantaneous sensory data, in such a way as to provide estimates of the robot's pose within its map: Guibas et al. [GMR] present the localization problem as a matching problem between a world polygon and a visibility polygon. The visibility polygon represents the output of a swept rangefinder. This algorithm provides a list of poses within the world polygon from which the view is consistent with the rangefinder polygon. We attempt in this paper to present algorithms that solve a similar problem, but one which lends itself somewhat better to the sensors and computational resources available on an actual mobile robot.

2.2. *Localization Algorithms*. The main results for this section comprise algorithms to compute *feasible pose sets*. A feasible pose set is the set of all poses in the robot's configuration space, C, which are feasible with the given inputs. Primarily, we consider

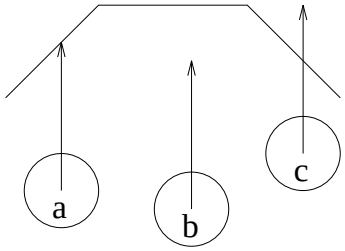


Fig. 4. A range probe is a vector with its tail at the robot and its head at an obstacle boundary. Pose (a) is feasible under the shown range vector, but poses (b) and (c) are not.

the case where C is $\mathbb{R}^2$, and the robot has perfect orientation sensing, but no position sensing. We also consider an $\mathbb{R}^2 \times S^1$ configuration space, where the robot has no knowledge or only limited knowledge about its orientation with respect to the world. In this section we will give the foreground of combinatorially precise computational geometric algorithms; we will discuss versions of these algorithms that can be made to run efficiently in Section 2.3.

2.2.1. *Definition of the Localization Operation*: *Computing Feasible Poses*. We represent a range reading as a vector with the tail at the robot. In order for a given position within a geometric map to be consistent with a particular range reading, the following statement must be true: If we place the tail of the range vector at the given position, then (i) the head of the range vector must lie on an obstacle but (ii) the body of the vector must not intersect any obstacles. This comes from the following interpretation of range reading: "a reading of 57 cm in direction θ means that there is nothing *within* 57 cm of the rangefinder in direction θ, but there is something *at* 57 cm." Possible positions of the robot, given the positions of the obstacles, meet the following condition: any place the tail of the vector can be positioned such that the head of the vector intersects an obstacle but no part of the body of the vector lies on an obstacle, represents a place where the robot can be and get that range reading. Figure 4 demonstrates this interpretation. Given a range vector z, then we can compute the possible locations of the robot. Before we can define feasible pose precisely, we need a few definitions: The Minkowski sum of two sets, $X \oplus Y$ (as described in [LP]) is the vector sum of all points in X with all points in Y:

$$X \oplus Y = \{x + y \mid x \in X,\ y \in Y\}.$$

$X \ominus Y$ is the vector sum of X with the 180° rotation of Y:

$$X \ominus Y = \{x - y \mid x \in X,\ y \in Y\}.$$

$\oplus$ is also called *convolution*; the Minkowski sum is mathematically closely related to the convolution integral used in signal processing. If x and y are points, $\ell(x, y)$ denotes the open line segment between (but not including) x and y. Let ∂X denote the boundary of the set X. Let M be a polygon representing the boundary between free space and obstacle in the robot's map. Let z be the vector representing a given range probe. We can

express the feasible poses for model M and probe z:

$$FP(M, z) = \partial(M \ominus z) - (M \ominus \ell(0, z)), \tag{1}$$

where "$-$"denotes set difference. In fact, we can dispense with the ∂, because the interior of $M \ominus z$ is contained in $M \ominus \ell(0, z)$. This gives us

$$FP(M, z) = (M \ominus z) - (M \ominus \ell(0, z)), \tag{2}$$

which is an equivalent definition, but that extends more easily to the case with uncertainty. For purposes of explication, we define two more sets, $D(M, z)$ and $E(M, z)$:

$$\begin{aligned} D(M, z) &= M \ominus z, \\ E(M, z) &= M \ominus \ell(0, z). \end{aligned}$$

Since $FP(M, z) = D(M, z) - E(M, z)$, $D(M, z)$ is, in some sense, the set of points "added in" by the head of a given range probe, while $E(M, z)$ is the set of points "subtracted out" by its body. Figure 5 illustrates an example calculation of $FP(M, z)$. Precise localization will, in general, take multiple range probes. Given a set of range probes Z, we define

$$FP(M, Z) = \bigcap_{z \in Z} FP(M, z). \tag{3}$$

We deal with translational uncertainty by one of two mechanisms: the *consistent reading* method and the *varying epsilon* method.

The *consistent reading* method says, "if $FP(M, Z)$ is empty (there are no points in the intersection of the results of all range readings), what is the set of all poses that are contained in a maximal number of $FP(M, z)$, for $z \in Z$?" For example, suppose there are 24 range readings taken (say, one every 15°), and no point in the map is in all 24 feasible pose sets, then look for the set of poses contained in any 23 of the sets, or any 22, and so on (this can be done without increased complexity, as explained below). This introduces a quality measure to the result: a point that is contained in all 24 point sets is a better match than one that is only contained in 23, or 22. This tactic enables the algorithm to be robust in the presence of bad range readings. Note that, in this context, *Bad* range readings are those that are missing or impossible, or which simply do not correspond to the actual distance from the sensor to the nearest object in the appropriate direction—essentially, sensor values that are outside tolerance bounds. It also can handle the case where a movable obstacle in the robot's vicinity occludes a stationary obstacle, causing one or more range readings to be different than expected (see, for example, Figure 6). This idea can be formalized as follows: If a set X is a subset of ambient space $\mathcal{U}$, then the *characteristic function* for X is a function $\chi_X : \mathcal{U} \to \{0, 1\}$ such that, for all $u \in \mathcal{U}$,

$$\chi_X(u) = \begin{cases} 1 & \text{if} \quad u \in X, \\ 0 & \text{otherwise.} \end{cases}$$

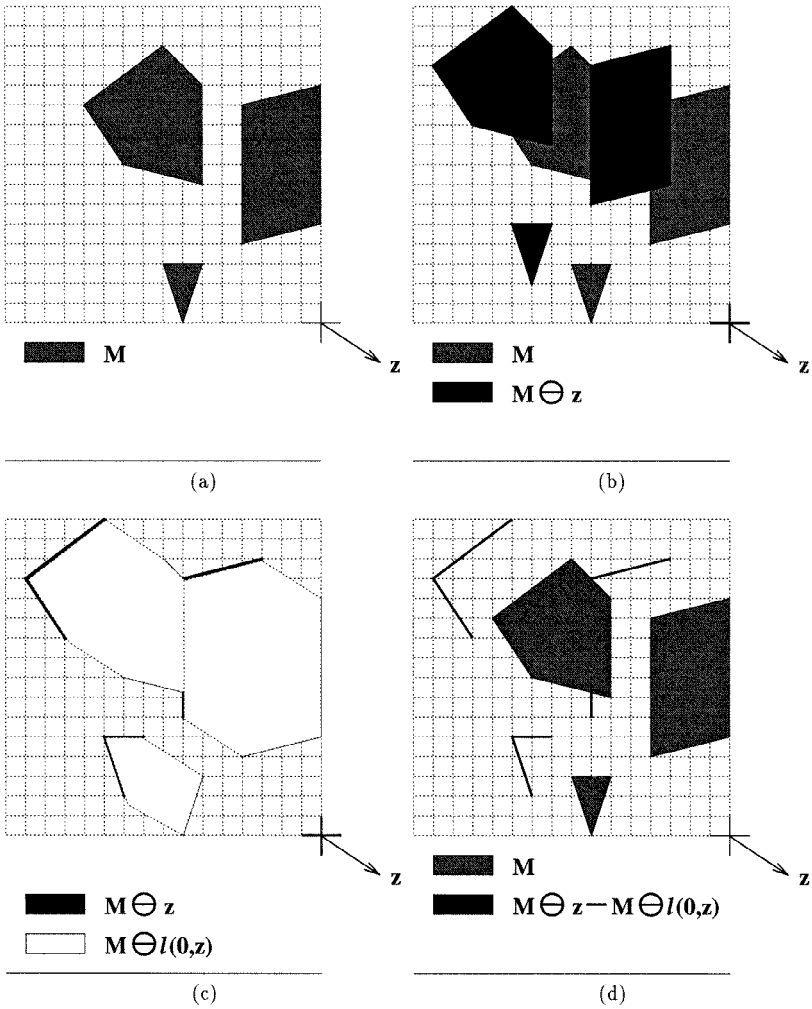


Fig. 5. (a) A set of polygons, M, and a range vector z. (b) M with $D(M, z)$ overlaid. (c) $D(M, z)$ with $E(M, z)$ overlaid. (d) $FP(M, z) = D(M, z) - E(M, z)$ shown with the original M.

Using this notation, the function that maps a point $p \in \mathrm{C}$ onto $\{0, 1\}$ based on whether $m \in FP(M, z)$ is $\chi_{FP(M,z)}$. Given a set Z of m range readings, we define $\mathrm{rnk}(M, Z, p)$ as the function from C to $\{0, 1, \ldots, m\}$ that, for each $p \in \mathrm{C}$, counts how many $FP(M, z)$ contain it. Precisely,

$$\forall p \in C, \qquad \mathrm{rnk}(M, Z, p) = \sum_{z \in Z} \chi_{FP(M,z)}(p).$$

We can now define the set of points in C that are contained in a maximal number of

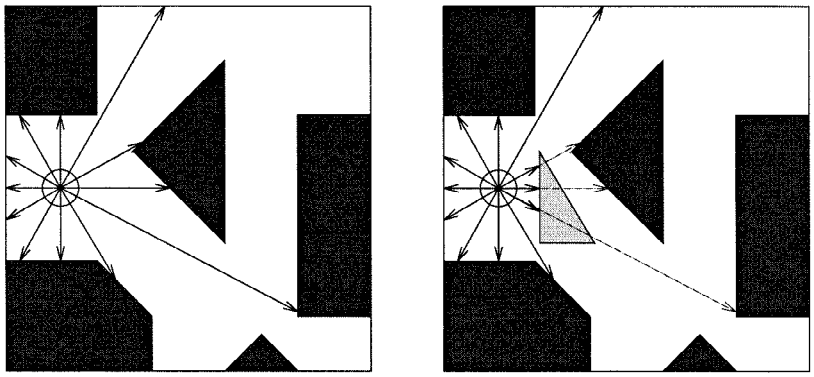


Fig. 6. The consistent readings method can handle new obstacles: if the robot (the circle) has a feasible pose under the twelve range probes shown in (a), it will still have a feasible pose meeting nine of the twelve range probes, when three of the range probes are changed by the introduction of a new obstacle.

$FP(M, z)$. Let $\kappa = \sup_{p \in C}(M, Z, \cdot)$. Then define

$$\mathrm{loc}(M, Z) = \{p \mid \mathrm{rnk}(M, Z, p) = \kappa\}.$$

The *varying-epsilon* method for coping with uncertainty uses epsilon balls (we refer to an uncertainty ball about the origin with radius ε as B_ε. B_ε will be an L_2 disk (Euclidean distance metric) unless otherwise specified). Given a map, M, and a range reading $z \in \mathbb{R}^2$, we want all poses p where $p \oplus z \oplus B_\varepsilon$ intersects an obstacle, but the portion of $\ell(p, p \oplus z)$ lying outside $p \oplus z \oplus B_\varepsilon$ does not. We denote the feasible poses for M and z with an uncertainty ball of radius ε by $FP_\varepsilon(M, z)$:

$$FP_\varepsilon(M, z) = (M \ominus (z \oplus B_\varepsilon)) - \left(M \ominus \ell\left(0, z\left(1 - \frac{\varepsilon}{\|z\|}\right)\right)\right). \tag{4}$$

Again, we define the "additive" and "subtractive" parts of the formula:

$$\begin{aligned} D_\varepsilon(M, z) &= M \ominus z \oplus B_\varepsilon, \\ E_\varepsilon(M, z) &= M \ominus \ell\left(0, z\left(1 - \frac{\varepsilon}{\|z\|}\right)\right), \end{aligned}$$

so that $FP_\varepsilon(M, z) = D_\varepsilon(M, z) - E_\varepsilon(M, z)$. We can now treat the range reading as having the form, say, "57 cm plus or minus 2 cm." While $FP(M, z)$ are typically sets of line segments, $FP_\varepsilon(M, z)$ is typically a set of two-dimensional "bands" that are, in effect, the line segments "grown" by ε. The intersection of two or more $FP_\varepsilon(M, z_i)$ is also typically a set of two-dimensional regions, rather than a set of points or segments. We now measure the quality of a match by saying "how large must we make ε in order

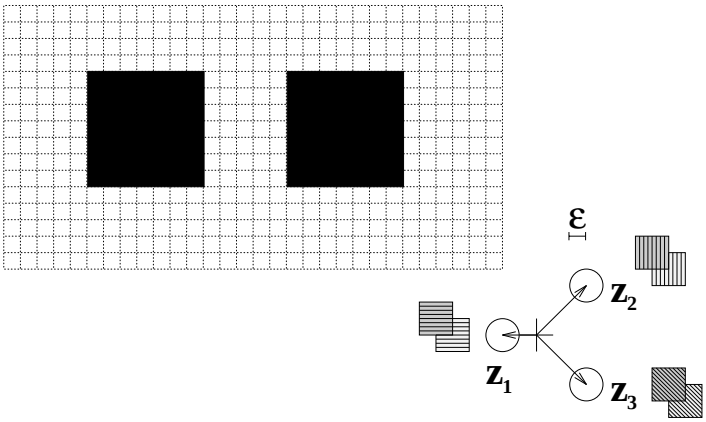


Fig. 7. A localization example: A map (the black squares), three range probes ($z1$, $z2$, and $z3$), and an uncertainty ball of radius ε. For Figures 7–11, the small, partially occluded grey squares key the shading of $D(M, z_i)$. The darker, unoccluded grey squares key the shading of $E(M, z_i)$.

to find a point that is in every feasible pose set?" We can also *combine* the consistent reading method and the varying epsilon method by asking "How large do we need to make epsilon so that there exists a point that is in, say, 90% of the feasible pose sets?" (If we have m range probes, what is the minimum epsilon such that there exists a point that is contained within $\geq 0.9m$ of the $FP_\varepsilon(M, z)$?) Figures 7–11 show an example of localization with three range probes. The first figure depicts the map, M, and the three uncertain range probes, z_1 through z_3; the next three show the D_ε and E_ε sets for the three range probes. The fifth figure shows $FP_\varepsilon(M, Z)$, the region that is consistent with all three range probes, overlaid with the original M and the three D_ε.

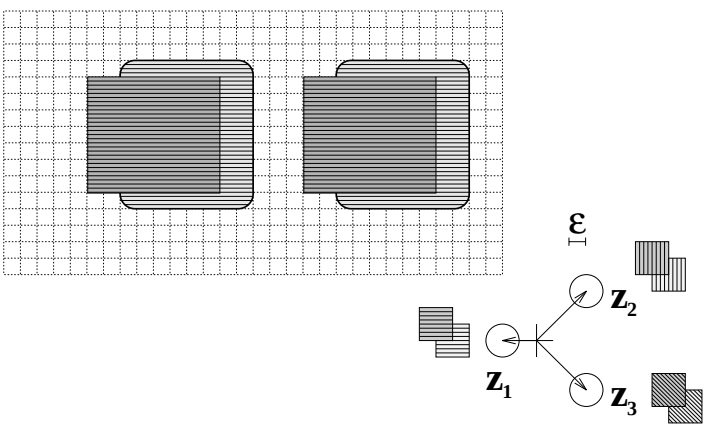


Fig. 8. $D(M, z_1)$ and $E(M, z_1)$.

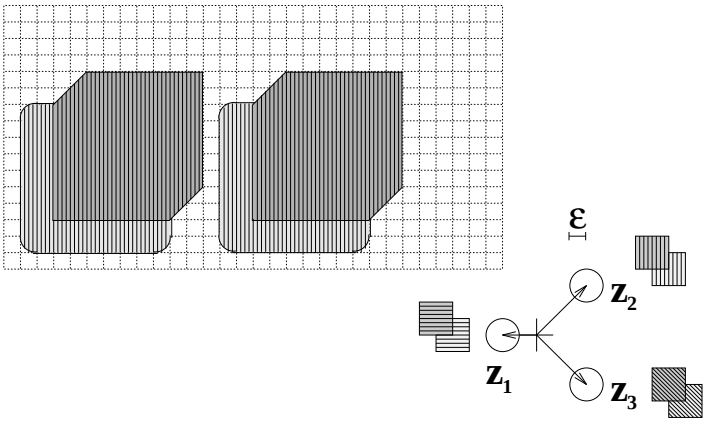


Fig. 9. $D(M, z_2)$ and $E(M, z_2)$.

2.2.2. *Exact Algorithms for Localization in* $\mathbb{R}^2$. Given localization operations that we have just defined, we can now define algorithms to compute the various feasible pose sets. Our first set of algorithms are exact combinatorial algorithms to perform localization when the map is a set of polygons in the plane. Initially, we define the geometric framework for these exact algorithms and provide a rough sketch of a brute-force algorithm to perform the localization operation. We then describe how we can use a "plane-sweep" technique to provide a localization algorithm with better asymptotic behavior than that of the brute-force method.

Equation (2) said that for map M and probe z:

$$FP(M, z) = M \ominus z - M \ominus \ell(0, z).$$

This is the set-theoretic definition of the feasible pose set for a given map and probe.

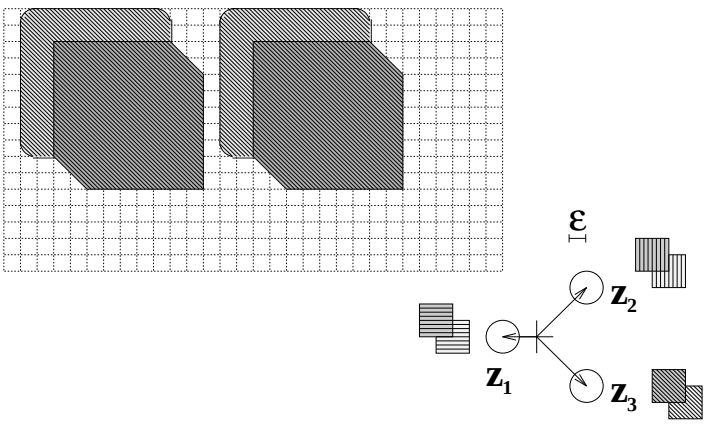


Fig. 10. $D(M, z_3)$ and $E(M, z_3)$.

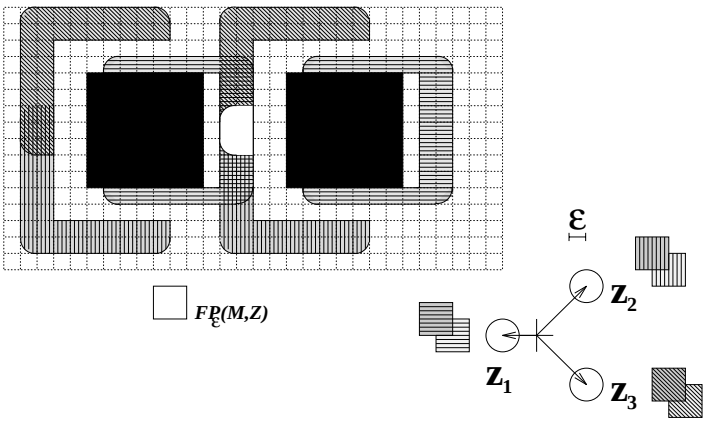


Fig. 11. Here, we show the three $FP(M, z_i) = D(M, z_i) - E(M, z_i)$ overlaid. The white rectangle with two curved corners in the center is the intersection, $FP(M, \{z1, z2, z3\})$.

We can develop an algorithm for computing feasible pose sets directly from this. Suppose M is a set of convex polygons with n vertices and edges. We can compute $D = M \ominus z$ in time $O(n)$. We can also compute $E = M \ominus \ell(0, z)$ in linear time: Lozano-Pérez [LP] showed that $B \ominus A$, for convex polygons B and A with b and a vertices, respectively, can be computed in time $\Theta(a + b)$. This time bound is achieved by building the result set directly by merging the edge lists of A and B in order of increasing orientation relative to the coordinate system. If we treat the closure of $\ell(0, z)$ as the boundary of a two-sided polygon, then we can apply this result to our problem. Note that D is simply M translated by $-z$, and is therefore still a set of polygons. E is a set of polygonal regions, but each region is an open set. Taking the set difference can be done in a naïve fashion in $O(n^2)$ time by taking the set difference between each region P_+ in D and each region P_- in E, yielding exactly $FP(M, z)$. Note that it is important in the exact case that we treat the regions in E as open, since the boundary of a region in E includes the boundary of the corresponding region in D (this follows from the definition), so if we do not treat the E objects as open sets, we will obtain no feasible poses.

We can adapt this algorithm to the varying-epsilon method in a straightforward fashion: D_ε is computed by convolving M with $z \oplus B_\varepsilon$, E_ε by convolving M with $\ell\,(0, z\,(1 - \varepsilon/\|z\|))$, as per (4). The cost of computing D_ε and E_ε is still linear. The brute-force algorithm we just described for the zero-uncertainty case can still be used here, except for two changes. First, we no longer have to worry about the boundaries of the E objects, and can treat all regions in both D and E as compact. Second, if our uncertainty region is an L_2 disk, then we have to allow for circular arcs in the boundaries of our objects. While this higher algebraic complexity creates additional implementation details, it has no effect on asymptotic combinatorial cost.

For a set Z of m different range probes, we can compute $FP_\varepsilon(M, Z)$ straightforwardly by first computing each $FP_\varepsilon(M, z)$ and then computing the intersection of those m sets. In general, a set of k linear and circular segments in the plane can have $O(k^2)$ intersections. Thus, if M has n edges, each $FP_\varepsilon(M, z)$ has complexity $O(n^2)$, so the union of all of the $FP_\varepsilon(M, z)$ has complexity $O(mn^2)$. Hence, we can compute the intersections of the

union in time[4] $O(n^4m^2)$. A property of each individual $FP(M, z)$, however, yields a tighter bound. Remember that the complexity of U is the number of edges and vertices required to describe it, while the complexity of an arrangement $\mathcal{A}$ is the number of vertices required to describe *it*. This complexity ($\mathcal{A}$'s) is bounded from above by $O(N+s)$, where N is the complexity of the generating sets, and s is the number of intersections between elements of these sets. In our case, $N = O(mn^2)$ because $FP(M, z)$ has complexity $O(n^2)$ and there are m z's:

LEMMA 1. *Suppose Z contains m range probes. Let U be the union*

$$\bigcup_{z \in Z} FP_\varepsilon(M, z).$$

Then (i) *the complexity of U is $O(mn^2)$, and* (ii) *the number of intersections in the arrangement $\mathcal{A}$ generated by the sets $\{FP_\varepsilon(M, z)\}_{z \in Z}$ is $O(m^2n^2)$. The number of intersections in $\mathcal{A}$ is $O(m^2n^2)$.*

PROOF. We have just shown (i) if M has n edges, each $FP_\varepsilon(M, z)$ has complexity $O(n^2)$, so the union of all of the $FP_\varepsilon(M, z)$ has complexity $O(mn^2)$. (ii) Although there are $O(n^2)$ segments on the boundary of an individual $FP_\varepsilon(M, z)$, they all lie on $\partial(D_\varepsilon(M, z))$, which is just a translation of the boundary of $M \oplus B_\varepsilon$. The segments, therefore, lie on $O(n)$ lines and circles. In addition, the interiors of these segments do not overlap. Thus, there can only be $O(n^2)$ intersections between them. Analogously, when we take the union of m $FP_\varepsilon(M, z)$, the $O(mn^2)$ segments lie on $O(mn)$ lines. $\mathcal{A}$, therefore, can only have $O(m^2n^2)$ intersections. □

In fact, these bounds are tight:

LEMMA 2. (i) *The complexity of U is $\Theta(mn^2)$, and* (ii) *the number of intersections in the arrangement $\mathcal{A}$ is $\Theta(m^2n^2)$.*

PROOF. (i) Figure 12 shows a set of polygons, M, which has n edges. It also shows $FP(M, z)$ for a sample range probe, z. If M has $n/8$ vertical rectangles and $n/8$ horizontal rectangles, there are $\Omega(n^2)$ of the open squares between the rectangles. Each open square contains part of $FP(M, z)$, so $FP(M, z)$ must have size $\Omega(n^2)$. $FP_\varepsilon(M, z)$ for ε smaller than half the size of the open squares, has similar appearance, but with regions such as shown in Figure 13. (ii) If we have a set of m different range probes, Z, such as the ones shown in Figure 13, with each $z \in Z$ shorter than the size of those open squares, each $FP_\varepsilon(M, z)$ will have that same complexity. U, therefore, for this M and Z, has $\Omega(mn^2)$ edges. Furthermore, we can select Z such that each $FP_\varepsilon(M, z)$ intersects every other within each of the open squares. In that case, each square contains $\Omega(m^2)$ intersections, leading to $\Omega(m^2n^2)$ intersections overall. Lemma 1 gives us upper bounds equal to these lower bounds, allowing us to conclude that these bounds are Θ-tight. □

[4] A more efficient algorithm is given later, in Theorem 7.

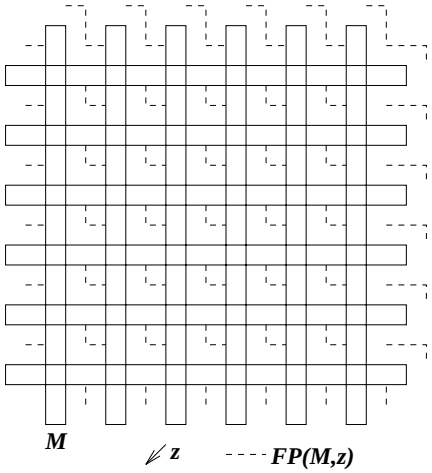


Fig. 12. A lower bound: M has n edges, but $FP(M, z)$ has $O(n^2)$ connected components.

Plane-Sweep Localization Algorithms. We now explain how to construct plane-sweep algorithms to compute the various types of feasible pose sets we have defined and to analyze these plane sweeps to determine their asymptotic behavior. The Appendix provides a review of the basic plane-sweep algorithm for finding all intersections between line segments in the plane and of the modification of this algorithm to output depth of coverage of an arrangement of polygons. The Appendix is a condensed treatment of

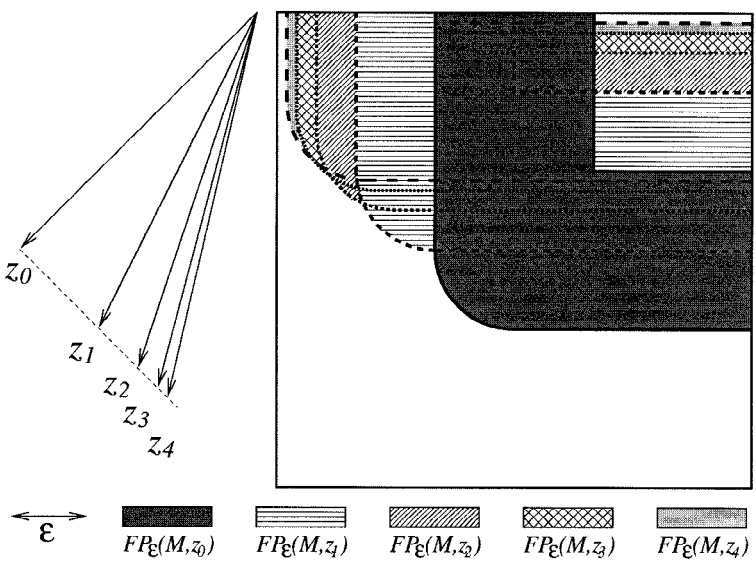


Fig. 13. A set of m range probes, $\{z_i\}$ which, together with a square region boundary, induce an $FP_\varepsilon(M, z)$ with m^2 intersections. The sequence of z_i have endpoints lying along a diagonal line, with the distance between adjacent z_i decreasing by a factor of two each time, allowing m to be arbitrarily large.

material found in, among other places, [PS], [NP], [Don], and [Lat]. The last paragraph of the Appendix describes a version of the polygon arrangement algorithm which outputs the depth of coverage of polygons of two different colors (e.g., how many *red* polygons cover a given member of the arrangement and also how many *blue* cover that member). The localization algorithm described below builds on the ability to compute polygon arrangements among polygons of multiple colors using a plane-sweep algorithm.

THEOREM 3. *Given a set of polygons M and a range vector z, we can compute $FP(M, z)$ in time $O((n + s)\log n)$, where n is the number of edges in M and s is the number of intersections between M and $M \ominus z$.*

PROOF. We can compute $D(M, z)$ and the union of $E(M, z)$ with its boundary each in time $O(n)$. The standard arrangement-computing plane-sweep algorithm runs on this collection of polygons (which has a total of $2n$ edges) in time $O((n + s)\log n)$, as shown in [NP]. We modify the sweep-line data structure so that each segment on the sweep-line knows how many polygons of $D(M, z)$ (color these polygons *red*) and how many polygons of $\partial(E(M, z))$ (color these polygons *blue*) cover it. Since we have a constant number of colors to keep track of, we can make this change without altering the asymptotic cost of processing each event. As we build up chains that, upon closing, are output as regions of the arrangement, we decorate each edge with the *red* and *blue* coverage depths of the region on each side of that edge; this information gets output with the edge list for each region. Once the arrangement has been computed, we make a list of all the edges (order is unimportant). This list is $O(n + s)$ long. We go through this list and output all of the edges that have *red* coverage of depth > 0 and *blue* coverage of depth > 0 on one side and *red* and *blue* coverages of depth 0 on the other. These are the segments of $D(M, z)$ that are not covered by the *interior* of any polygon of $E(M, z)$, and that therefore lie in $D(M, z) - E(M, z)$. This step takes time proportional to the length of the list to perform. The overall work done therefore to compute the annotated arrangement and filter out the segments that are part of $FP(M, z)$ takes time $O((n + s)\log n)$. □

THEOREM 4. *Given a set of polygons M, a range vector z, and an uncertainty value ε, we can compute $FP_\varepsilon(M, z)$ in time $O((n + s)\log n)$.*

PROOF. As noted previously, $FP_\varepsilon(M, z)$, for $\varepsilon \neq 0$ is a set of generalized polygons (line segments and circular arc segments). The introduction of circular arc segments does not change the asymptotic behavior of the plane-sweep algorithm. For this problem, we again use the arrangement-generating plane sweep with two colors, red and blue. Since the feasible poses are now, in general, regions of nonzero area, we only need to output for each region what its *red* and *blue* coverage depths are. We then filter the list of output regions, keeping all regions with $(red, blue)$ coverage of $(k, 0)$, for any $k > 0$. In this case the filtering step takes time proportional to the number of regions, which is bounded by the number of edges in the arrangement. Since both the arrangement computation and the filtering computation are no more complex than in the zero-uncertainty case, we conclude that we can compute $FP_\varepsilon(M, z)$ in time $O((n + s)\log n)$. □

The analysis for multiple range probes goes similarly. The main difference in the analysis is that we must establish that we can perform all the work necessary to keep track of $2m$ colors without increasing the asymptotic cost of the algorithm. Fortunately, we do not have to perform a general $2m$-colored arrangement, Theorem 5 explains how Lemma 1 allows us to extend Theorems 3 and 4 efficiently to the computation of feasible poses with multiple range probes:

THEOREM 5. *Given a set of polygons, M and a set of range probes, Z, we can compute $FP(M, Z)$ in time $O((mn^2 + s)\log(mn))$, where n is the number of edges in M, m is the number of range probes, and s is the number of intersections in the arrangement generated by the sets*

$$\{D(M, z), E(M, \ell(0, z))\}_{z \in Z}.$$

Furthermore, s is $O(m^2n^2)$ in the worst case.

PROOF. If we can get each $FP(M, z)$ to be nonoverlapping (no two regions intersect), then we can compute $FP(M, Z)$ by computing the intersection of the m $FP(M, z_i)$ by a straightforward arrangement algorithm: we run a plane-sweep algorithm over the union of the $FP(M, z_i)$ and extract the regions with coverage depth m. Each of the $FP(M, z_i)$ can have complexity $O(n^2)$, so we would expect computing the arrangement of the m $FP(M, z_i)$ to take time $O(m^2n^4\log(mn))$. Lemma 1, however, says that this arrangement will only have $O(m^2n^2)$ intersections; so its computation will take time only $O(m^2n^2\log(mn))$ in the worst case. Therefore, we can use the (conceptually simple) method of taking the m individual arrangements and computing the intersection of those, and still meet our time bound. □

COROLLARY 6. *We can compute* $\mathrm{loc}(M, Z)$ *in time* $O((mn^2 + s)\log(mn))$.

PROOF. Since we have to examine each region boundary of the arrangement in any event, we can count the number of range probes each region boundary is consistent with at no extra asymptotic cost, since addition is no more expensive than ANDing. By outputting coverage information with each edge or vertex, we increase the size of the arrangement by a factor of $O(\log m)$, but that leaves the time to output the arrangement no longer than the time to compute it. Thus, we can output the count-annotated arrangement in the same time as we can output the coverage-annotated arrangement. □

THEOREM 7. *Given a set of polygons, M, a set of range probes, Z, and an uncertainty value ε, we can compute $FP_\varepsilon(M, Z)$ in time $O((mn^2+s)\log(mn))$, where n is the number of edges in M, m is the number of range probes, and s is the number of intersections in the arrangement generated by the sets*

$$\{D_\varepsilon(M, z), E_\varepsilon(M, \ell(0, z))\}_{z \in Z}$$

(which is $O(m^2n^2)$ in the worst case).

COROLLARY 8. *We can compute* $\mathrm{loc}_\varepsilon(M, Z)$ *in time* $O((mn^2 + s)\log(mn))$.

Theorem 7 follows from Theorem 4 and Lemma 1 in the same way that Theorem 5 does. Corollary 8 follows from Theorem 7 using the same arguments used to get from Theorem 5 to Corollary 6.

Finally, we wish to consider the problem of determining the minimum uncertainty, ε, such that $FP_\varepsilon(M, Z)$ is nonempty. Megiddo, in [Meg], provides a mechanism for transforming decision procedures into minimization procedures, for some problems. This mechanism, known as *parametric search*, applies to decision procedures that have a distinguished quantitative input, d (for example, a decision procedure $P(A, B, d)$ that determines whether $f(a, b) \leq d$, for a given function f of two arguments, A and B), such that there exists a critical value d^* for any instance of A, and B, such that for $d < d^*$, $P(A, B, d)$ is "*no*," and for $d \geq d^*$, $P(A, B, d)$ is "*yes*." The minimization problem $P_{\min}(A, B)$ is to find the value of d^*. Parametric search typically takes a serial decision procedure that operates in time $O(T(n))$ and a parallel decision algorithm that operates in parallel time $O(P(n))$, and produces a minimization procedure that operates in time $O(T(n)\log^k(P(n)))$ for some positive integer k. Agarwal et al. [AST] show how this mechanism can be applied to finding the minimum hausdorff distance [HK1] between two point sets. We can use a similar approach to develop an algorithm to compute the smallest uncertainty ball, $\varepsilon_{\min}$, such that $FP_{\varepsilon_{\min}}(M, Z)$ is nonempty. Agarwal et al. [AST] take a decision procedure with worst-case time $O(m^2n^2\log(mn))$ and are able to obtain a minimization procedure that takes worst-case time $O(m^2n^2\log^3(mn))$. We conjecture that we can do likewise with our problem.

2.2.3. *Allowing Rotational Uncertainty.* Until now, we have assumed that the robot has perfect orientation information; i.e., that it always correctly knows which direction it is facing. This may not be a valid assumption. Even if the robot has a compass, that compass will have only limited resolution and accuracy; the error may not be small enough for us to ignore it. If robot does not have a compass, then it is reduced to relying on its orientational odometry. Even if the orientational odometry is very good, there will arise occasions, such as initialization-time or after a sliding motion, when we will need to determine the robot's orientation by comparing sensory data with previously gathered information. For experiments performed to date using our robots, we have been able to use orientational odometry with fair success; however, robots with less fortunate kinematics may not permit this. For example, see the description of the treaded robot CAMEL in [Bro].

It is probable that we could perform localization in $\mathbb{R}^2 \times S^1$ by computing feasible pose sets in $\mathbb{R}^2 \times S^1$ directly. The planar map used thus far in the paper is essentially the configuration space of a point robot moving among planar obstacles (for a more detailed discussion of configuration spaces, see [LP] or [Lat]). The configuration space of a point robot with orientation is essentially the map we would use for $\mathbb{R}^2 \times S^1$ localization. By convolving that three-dimensional set with a set of range probes, we can formulate D and E sets analogous to the ones computed in the plane. We can use a space-sweep algorithm (the three-dimensional analog to a plane-sweep algorithm) to compute these feasible pose sets. We have not fully explored the detailed analyses or the complexity bounds for the sweep algorithm needed for this particular problem; however, we believe the cost to be high (on the order of $O((m+n)^6\log^k(mn))$ for some k), as the complexity of the problem seems closely related to that of finding the minimum Hausdorff distance between two

sets of points, segments, and polygons in the plane under Euclidean motion (translation and rotation). The current best algorithm for this problem (proposed in [CGH$^+$]) has an upper time bound of $O(m^3n^3\log^2(mn))$, for two sets P with complexity m and Q with complexity n. This bound appears to be close to optimal (within $O(\log^k(mn))$) [Ruc].

Theta-Slice Approximating $\mathbb{R}^2 \times S^1$ Localization. Perhaps a more practical method of handling rotational uncertainty in the localization process is to approximate the procedure just outlined by performing translation-only localization at a set of discrete rotations. We can compute M_{k_r} for $k = 0, 1, \ldots, p-1$ in $O(np)$ time, where M_{k_r} is M rotated by $2\pi k/p$. If we can perform the translation-only localization at each $k\Delta\theta$ in time $T(n, m)$ for an environment M of size n and m range probes Z, then we can determine which of the $k\Delta\theta$ angle offsets is (are) most compatible with M and Z in time $O(pT(n, m))$.

Separating Orientation Localization from Translation Localization. We may not want to extend the localization algorithm to handle rotational uncertainty at all. Under some robot and environment models, it makes more sense to perform rotational localization separately from translational localization. Among the models to which this applies are (i) robots equipped with bounded-error compasses or gyroscopes, and (ii) robots operating in environments such as uncluttered office environments where a large majority of the vertical surfaces in the environment have orientation along a set of cardinal axes. In cases like this it may make more sense to use some sort of sensory alignment procedure to reduce orientation uncertainty; using such a procedure in coordination with a translation-only localization algorithm may be more efficient than using an $\mathbb{R}^2 \times S^1$ localization algorithm.

2.3. *Using Rasterization for Practical Localization Algorithms.* For performance reasons, we use rasterized algorithms when we implement geometric algorithms on the robot. In our experience, combinatorially precise geometric algorithms such as those developed in Section 2.2 are not possible to implement on real robots. Rasterization is a necessary component of the algorithmic development, and as important as the design and analysis of the combinatorial algorithms. More strongly, we feel that any description of our work would be incomplete, if not misleading, were it not to describe the rasterization. To our view, such a lacuna would be as serious as omitting the asymptotic bounds. Rasterization is not a mere "engineering detail" but, rather, an integral part of any systematic attempt to develop physical geometric algorithms that are to be implemented and used in practice. See [DKM] for a methodology for the integrated codevelopment of combinatorially precise and rasterized geometric algorithms.

A *rasterized algorithm* employs the following technique: given a geometric description (for example, lists of edges or vertices), embed that description in an array by digitizing the primitives (in essence, "plotting" geometric primitives from the description on an initially blank bitmap). Given an algorithm that runs on geometric descriptions, convert (by hand) the algorithm to perform on a discretized grid. We believe that algorithms operating on discretized arrays are useful, in that they are often easier to implement than the original combinatorial algorithms, and they often run faster. See [Lat], [BL], [LP], [LRDG], [HK2], and [CK] for examples of how rasterization can be applied to robotics algorithms. In this section we define rasterized algorithms, and

define some basic building blocks from which rasterized algorithms can be built. We then present rasterized versions of the translation-only ($\mathbb{R}^2$) localization algorithms presented in Section 2.2.2.

Another reason for preferring rasterized algorithms for on-robot implementations lies in the type of data that is actually available to the robot. Many results in theoretical robotics are built on the data type *set of polygonal obstacles* (polyhedral obstacles, in three dimensions). It is impossible, however, to make a map that is a good polygonal/polyhedral representation of an unstructured environment. Part of this is due to limited sensor accuracy, but there are more fundamental problems: For one thing, the world is not made up of polyhedral objects. Objects in the world, such as piles of cables or hockey bags, can often only be represented approximately using polygonal/polyhedral shapes. The cost of representing such an object can be high, as it may require a large number of segments or faces. In addition, obtaining that representation will likely require arbitrary application of line- and curve-fitting procedures. Given a specification of required accuracy for our maps, we believe it is simpler to select a minimum feature size (based on that specification), and to make our maps using that feature size as their resolution. Thus, we feel rasterized maps are the most straightforward way to represent the real-space world our robot encounters. We want, therefore, to use rasterized computational geometry algorithms for manipulating our maps, since using standard computational geometry algorithms would require frequent conversion between rasterized and polygonal representations, leading to loss of accuracy.

2.3.1. *Introduction to Rasterized Computational Geometry.* Rasterized algorithms are not new. Many existing rasterized algorithms were originally developed by researchers in computer vision and computer graphics. Computer vision researchers often use rasterized algorithms, largely because computer vision algorithms operate, at least at low levels, upon images obtained from video cameras or scanners, which invariably *provide* rectangular arrays of intensity and/or color values. Computer graphics researchers design algorithms that create images for display upon video screens, which almost always *expect* rectangular arrays of intensity and/or color values. A video image has limited resolution, causing features below a certain scale to disappear and the locations of larger features to be known only approximately (usually to the nearest pixel). Thus, it makes sense to design computer vision and computer graphics algorithms that will operate well on rectangular arrays of limited resolution data. For example, there is often little advantage to a hidden-surface removal algorithm employing exact (rational) arithmetic and numbers for precision, if the output is to be displayed on a screen with 1024 pixel $\times$ 768 pixel resolution.

2.3.2. *Some Building Blocks for Rasterized Algorithms.* The fundamental data type of rasterized algorithms is the *grid*. A grid is divided up into *cells*, each of which can take on an integer value. Most operations fall into two categories: grid-level and cell-level. Basic grid-level operations include cellwise arithmetic and logic (for example, given two equal-sized grids, A and B, $(A + B)[i, j] = A[i, j] + B[i, j]$) and transformation operations (e.g., x–y-translation ($A[i, j] = B[i + p, k + q]$) and reflection ($A[i, j] = B[p - i, q - j]$)). Cell-level operations typically assign a value to each cell as a function of its neighbor cells. More complex functions are built up from sequential applications of these basic operations.

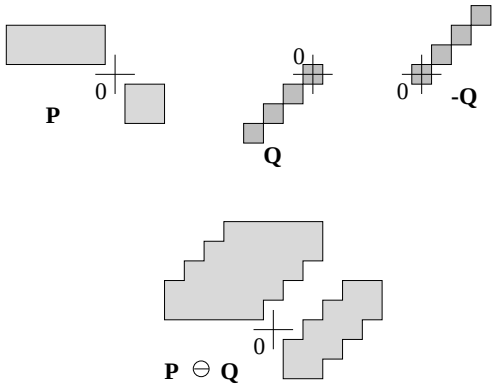


Fig. 14. Discretized convolution: $P \ominus Q$.

Discrete Convolution. The Minkowski sum operation, defined in Section 2.2.1 as

$$P \oplus Q = \{p + q \mid p \in P, q \in Q\}$$

can be applied directly in the rasterized domain. In the continuous domain, if P and Q are finite sets of discrete points, we can compute $P \oplus Q$ by a simple iteration: for each $p \in P$, iterate over all $q \in Q$, outputting $p + q$. In the rasterized domain there are no continuous point sets: curves and areas in the continuous domain are transformed into contiguous sets of pixels. Since all finite sets of discrete points have finite cardinality, we can compute $P \oplus Q$ for any pair of finite rasterized sets, P and Q. The operation denoted $\oplus$, applied to a rasterized domain, is exactly the operation that is known in digital signal processing [OW], [OS], image processing, and computer vision circles [MH], [Mar] as *convolution*. In digital filter design, for example, a common application of convolution is the Finite Impulse Response filter, whose output is computed by convolving a linear or rectangular region with a sampled input. For example, many of the "smoothing" filters and edge-detectors in common use today rely on this operation. Since it is very common, convolution is an operation that many processors (particularly those designed specifically for digital signal processing) are optimized to perform. Since specialized signal/image processing hardware is capable of performing this operation very quickly, discrete convolution is a very useful operation for rasterized-algorithms. Figure 14 shows a sample rasterized convolution.

Floodfill/Brushfire Algorithms. Breadth First Search [AHU] is an often-used order for visiting nodes in a graph, in which we visit/examine/compute upon the nodes adjacent to the start node first, then those that are at distance two from the start, then those at distance three, and so on until all nodes have been visited. In the context of rasterized algorithms, this order is known as a "floodfill" or "brushfire" algorithm. In its simplest form, the procedure to floodfill an array can be described as (i) number all source cells 0 and place them on the floodfill queue, and (ii) while the queue is nonempty, dequeue a cell c; if c bears the number i, number all of its unnumbered neighbor cells $i + 1$ and enqueue them. This algorithm floods from a set of source cells in contours that are determined by the definition of *neighbor*. If we define neighbors to be those cells that share a side

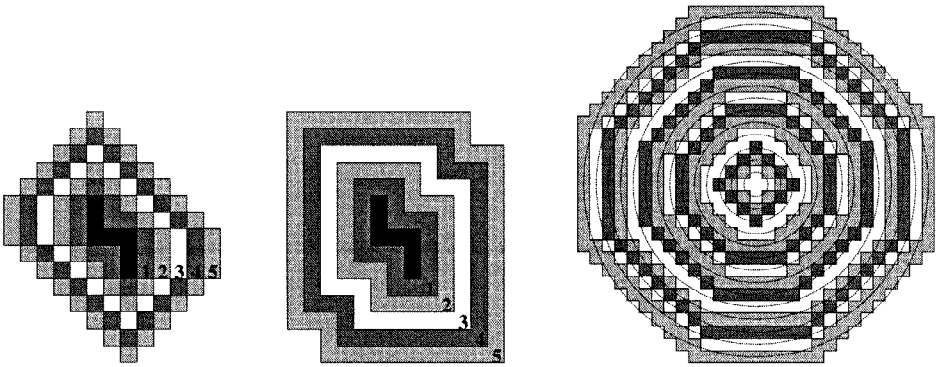


Fig. 15. (a) A pixelized shape shown with flooded L_1 and L_∞ contours. (b) L_{octagon} contours flooded out from a point source. L_2 circles are shown for reference.

(resp. side or corner) with the current cell, then we flood in diamond (square) shapes so that the number of a given cell is its distance to the nearest source cell under an L_1 (L_∞) distance metric. Figure 15(a) shows the shape of these contours for a sample pixelized shape.

We can define *obstacle cells* in the array as those that cannot be flooded. If we do this, then the contours define the distance to the nearest source cell via a path that does not cross any obstacle cell. In fact, in [LRDG] this property is used to design a very fast robot path planner that computes the shortest path from anywhere in the robot's configuration space to a specified goal. Once the contours are in place, the path from a given start point to that goal can be determined in real time, provided it exists; if there is no path to the goal, this can be determined virtually instantaneously. We use a variant of this planner on LILY.

Sometimes we want to be able to flood from a set of source points only out to a specified distance. For example, consider the C-space obstacles generated by a path-planner for a circular robot. These C-space obstacles are the real obstacles grown out by the diameter of the robot in all directions. Thus, if we are interested in the grown-obstacle-map of a diamond- or square-shaped (i.e., L_1 or L_∞ circular) robot with radius r, we can easily compute its C-space by the following modified floodfill algorithm: (i) number all obstacle cells 0 and place them on the floodfill queue, and (ii) while the queue is nonempty, dequeue a cell; *if* its number is less than r, number all of its unnumbered 4-connected neighbor cells (resp. 8-connected neighbor cells) one higher and enqueue them. This algorithm floods from a set of source cells in contours that are circular under an L_1 (resp. L_∞) distance metric. It only floods out to a specific distance, however, so that the set of all numbered cells after the algorithm is run is the same set of cells as obtained by taking the Minkowski sum of the obstacle map and an L_1 (L_∞) disk of the radius of the robot. Since we never enqueue a cell more than once, the cost is only proportional to the area of the map with the flood-fill case, as opposed to the map-area times disk-area cost of the Minkowski sum. Unfortunately, it is not possible to flood in this same fashion using an L_2 disk. It is, however, possible to approximate a small L_2

disk (one with radius less than 20 cells) with an octagonal region (an "L_{octagon}" disk) whose boundary is never more than one cell-size away from the boundary of an L_2 disk of the same radius. This "L_{octagon} disk" is obtained by alternating in a prespecified pattern between 4-neighbor and 8-neighbor growth. Figure 15(b) shows the distance contours associated with L_{octagon} growth, along with the L_2 circles they approximate. Which set of neighbors we use for a particular cell depends solely on the value assigned given to that cell. Given that we ordinarily use a map whose cell-size is within an order of magnitude of the robot size, this method of determining the grown-obstacle-space is adequate, and in practice much faster than taking the Minkowski sum.

It is also useful to be able to "flood" in a specific direction. A sample application here is computing the grown-obstacle-space of a rod-shaped robot at various orientations. As before, we could perform this operation by taking the Minkowski sum of the original grid contents and the pixelized vector representing the direction and distance of growth. A faster way to do this is to extract from the pixelized vector V_r a list of "neighbor numbers," which map from the pixels of V_r to the integer range $[0-7]$. *neighbor*$[i]$ represents the direction from the ith pixel of V_r to the $(i+1)$st (its neighbor going away from the origin): the pixel to the right has neighbor number 0; the pixel above and to the right has neighbor number 1; the rest of the neighbors are numbered in counterclockwise order from there. Now, when we do the floodfill, if we take a cell numbered i off of the queue, we only number and enqueue the one neighbor cell that is indicated by *neighbor*$[i]$. This way, we flood from source cells only in the direction of the pixelized vector. In fact, making this approach work properly is slightly more complicated: Since the grid really only recognizes eight distinct directions at a local level, it is possible for the vector that should stem from a source pixel to be partially or wholly omitted, in the case where the surface tangent of an obstacle at that source pixel is close to the direction of the range vector. This happens when the 8-directional pixelized vector points locally into the obstacle, even though the original vector points out of the obstacle. Figure 16(a) shows an example of this problem. The vector shown is sourced from the starred cell; the first three cells of the pixelized vector lie within the obstacle, even though the original vector points out of the obstacle when sourced from the obstacle's upper boundary. We address this difficulty using a lookahead scheme that causes the flood to work correctly without affecting the algorithm's asymptotic performance. In Figure 16(b), we have added the pixelized vector sourced at the cell marked "+." Hashed cells lie on the pixelized vectors sourced at either "$*$" or "+," while cross-hashed cells are those free-space cells that lie

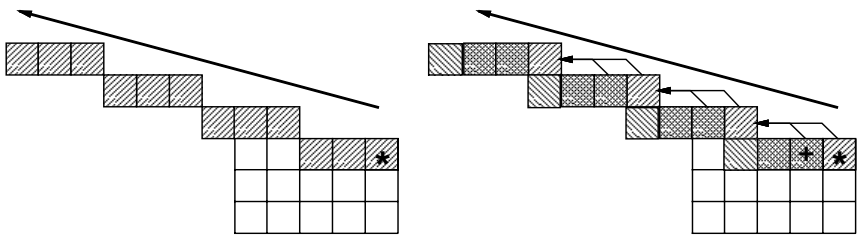


Fig. 16. (a) A pixelized vector may overlap a pixelized object, even though the original (unpixelized) vector points out of the object. (b) To avoid this and some other difficulties, we use a lookahead scheme based on the way the pixelized vector overlaps a translated copy of itself.

on both pixelized vectors. The cells that are forward-hashed, but not back-hashed are those that would not get numbered if the "$*$" vector were eliminated but the "$+$" vector remained. To ensure that these cells get numbered properly, at each step the floodfill procedure looks ahead to two cells: the cell indicated by the neighbor number of the current cell and the cell that is in the position of the nearest higher-numbered cell that *would not* get marked if the neighbor cell already bore a lower number. Which cell that is for a given potential can be precomputed once for each cell on the pixelized vector at the beginning of the fill, by examining how the pixelized vector overlaps a translated copy of itself (translated by one pixel in the dominant direction). Essentially, we create a picture like the one in Figure 16(b), for the particular vector being used, and look at the overlap.

The directional floodfill algorithm has an additional small performance advantage over omnidirectional floodfill procedures: Since we know in advance the direction in which growth will be taking place, it is not in fact necessary to use a queue-based implementation to perform directional flooding. It is sufficient to make a single ordered sweep across the grid, with the direction of the sweep determined by the orientation of the growth vector. In an omnidirectional fill, we must be certain that all cells at a distance less than k from the source cells have been processed before beginning to process the cells at distance k; otherwise, some cells may get a larger than necessary potential number (in fact, we can prevent this, but only by increasing the cost of the procedure due to the necessity of processing some cells multiple times). In the directional case, we only need to ensure that the lower-numbered cells that contribute to the numbering of a cell at distance k be numbered. For this reason, we can sweep the grid in a direction that will cause cells on a straight line from a given source cell to be processed in order of increasing distance. This does not affect the asymptotic behavior of the flooding procedure, but may allow significant reduction in the constants pertaining to the performance of a particular implementation of the procedure.

2.3.3. *Rasterized Localization in* $\mathbb{R}^2$

Localization Using Discrete Convolution. A simple implementation of the algorithm to compute the feasible pose set for a given rasterized range vector, map, and uncertainty follows directly from the definition of $FP_\varepsilon(M, z)$ in (4): suppose we have an inverted range vector z, uncertainty ε, and map M (Figure 17). Create a set Z_r that is the set of integer grid-points that intersect the vector z at a distance greater than ε from the head of z. Create also a set Z_ε that is the set of integer grid-points with a Euclidean distance of ε of the head of z. Regard the map M conceptually as a set of points, but maintain it as a two-dimensional array. Compute D (which can be regarded as the set of all cells that are within ε of a cell that is itself the vector sum of z with some obstacle cell) by convolving M with Z_ε. Remember that convolution of a bitmap with a point set is equivalent to offsetting the bitmap by the coordinates of each member of the point set, and computing the union of each of those offset bitmaps by ORing them all together. Next, we compute an intermediate set E (the set of all cells that are within $r - \varepsilon$ of an obstacle cell in the direction of z) by convolving M with Z_r. Now, FP is just the set difference $D - E$.

The complexity analysis is straightforward. Given a map with size $n \times n$, a vector that discretizes into r cells, and an uncertainty ball of radius ε, then the time required can be expressed $O(n^2(r+\varepsilon^2))$. This is the number of cells in the map times the number of times

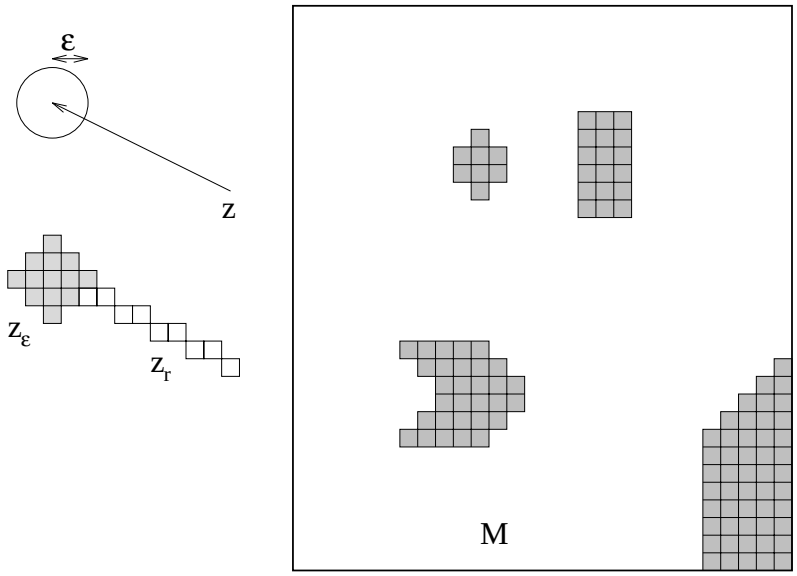


Fig. 17. A map, M, a range probe z, and an uncertainty ball of radius ε. z_r is the rasterized representation of $\ell(0, z(1 - \varepsilon/\|z\|))$. z_ε is the rasterized representation of $(z \oplus B_\varepsilon)$.

that an offset-and-copy operation has to be performed on the map. With multiple range probes, the cost is linear in the number of such probes: we can intersect the m feasible pose sets $FP(M, z_i)$ in time $O(n^2 m)$, which does not increase the asymptotic cost beyond the $O(n^2(r + \varepsilon^2)m)$ total cost for computing the m individual $FP(M, z_i)$. Determining the coverage of each cell in M can be done in the exact same time complexity by counting the number of $FP(M, z_i)$ that contain that cell instead of ANDing the cells together. In the discretized case we can determine the optimal ε for a given coverage at a cost increase of $O(\log n)$ by a simple binary search tactic: since we can only select integer values for ε, we need not resort to parametric search. This gives us an overall time complexity for determining the smallest nonempty $FP(M, Z)$ of $O(n^2 m \log n(r + \varepsilon^2))$.

Speeding up the Rasterized Localization Algorithm. The algorithm of Section 2.3.3 does redundant work. For a single range probe z, each cell in the map M is copied and operated upon $O(r + \varepsilon^2)$ times, for r the length of the range probe and ε the radius of the uncertainty ball. This is true even of those cells that are inside an obstacle, or out in the middle of free-space. We can reduce the cost of computing an individual $FP_\varepsilon(M, z)$ to $O(n^2)$, independent of r and ε. This means we can compute the smallest nonempty $FP_\varepsilon(M, Z)$ in time $O(n^2 m \log n)$.

We can reduce the cost of computing $FP_\varepsilon(M, Z)$ to $O(mn^2 r)$ by approximating the error ball so that we can use a floodfill-based method to compute D. We do this using much the same method as used to compute the C-space obstacles: We translate the map by $-z$, then grow it by an "L_{octagon}" disk of radius ε, using a limited range floodfill, as described in Section 2.3.2. This is the approximate D set, which is almost the same as the D computed by the more expensive convolution method. It is,

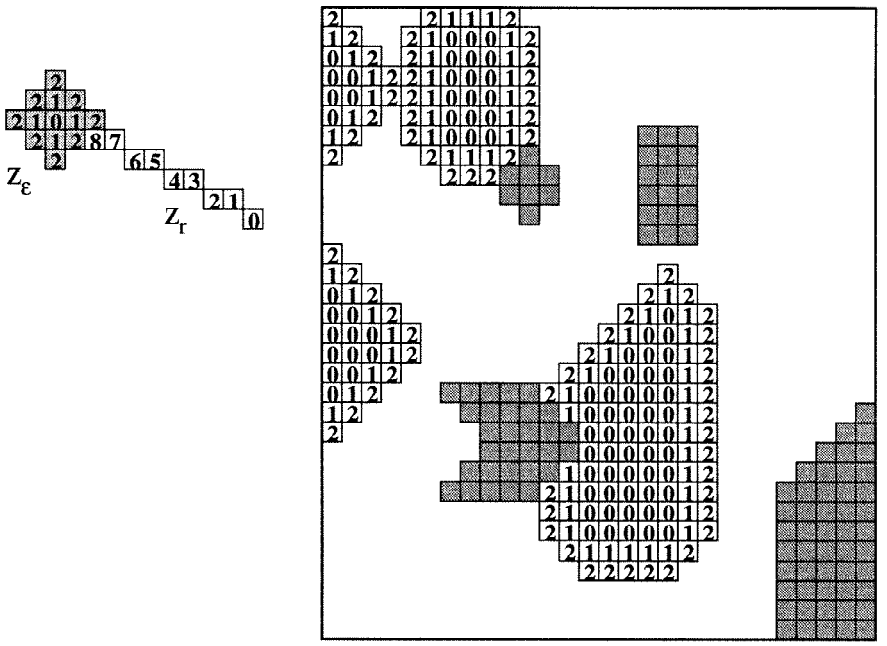


Fig. 18. $M \ominus z$ flooded by an ε of 2, shown with the original M.

in fact, identical for $\varepsilon \leq 4$ cells. For ε between 5 and about 20, the boundary of the L_{octagon} disk of radius ε lies within one cell-size of the L_2 disk of the same radius. Figure 18 shows a sample rasterized M along with the D set. Translated M pixels are numbered 0; pixels at distances 1 and 2 away from 0-numbered pixels are numbered 1 and 2.

We further reduce the cost to $O(mn^2)$ by using the directional floodfill of Section 2.3.2 to compute E. For a range vector at angle θ of length r (resp. $r-\varepsilon$, if the uncertainty value is nonzero), we do this in the following way: we start with the obstacle pixels in place and numbered zero, and perform a directional floodfill out to distance r (resp. $r-\varepsilon$). Finally, we compute $FP_\varepsilon(M, z)$ by merging the two arrays by the following algorithm: (i) Set all numbered cells in $D(M, z)$ to a distinguished negative value k_D. (ii) If a cell in $E(M, z)$ has a number, we copy that number into the corresponding cell in $D(M, z)$. All cells that still bear the value k_D are feasible pose cells for map M and range probe z. Figure 19 shows a directional flood from M in the direction of $-z$. Figure 20 shows $FP_\varepsilon(M, z)$: numbered cells are in $FP_\varepsilon(M, z)$. The white regions are cells covered by D_ε, and gray regions are the original M.

2.4. *Summary.* In this paper we defined a model of mobile robot localization that builds upon the concept of *feasible poses*, which are poses for a mobile robot that are consistent with available range and map data. We model a point-and-shoot rangefinder as a sensor that returns the distance to the nearest object in the direction of its sensitivity, toleranced by an error bound. A feasible pose relative to a map and a range vector is essentially a pose

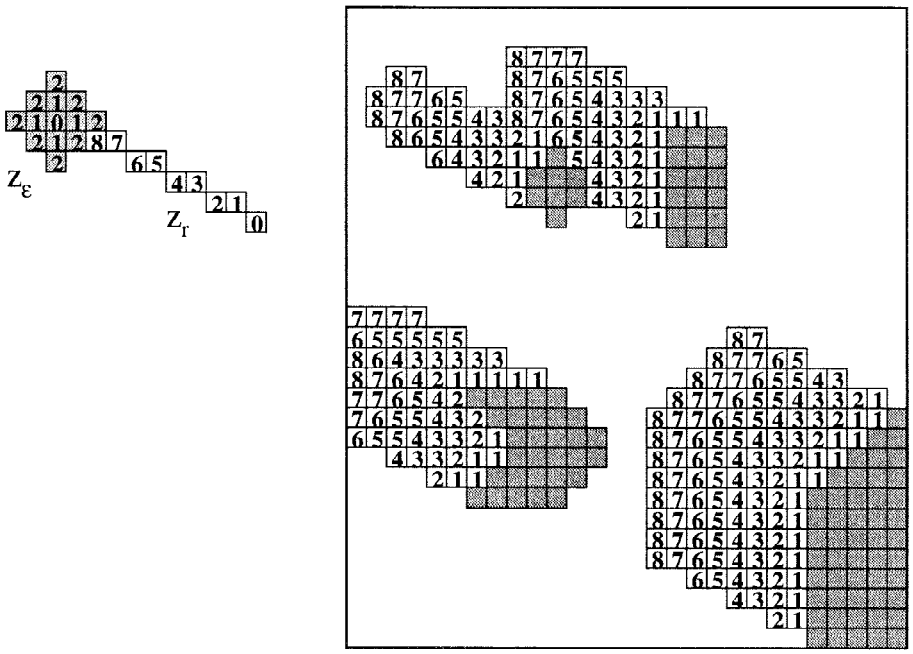


Fig. 19. M flooded in the direction of z by distance $||z|| - \varepsilon$.

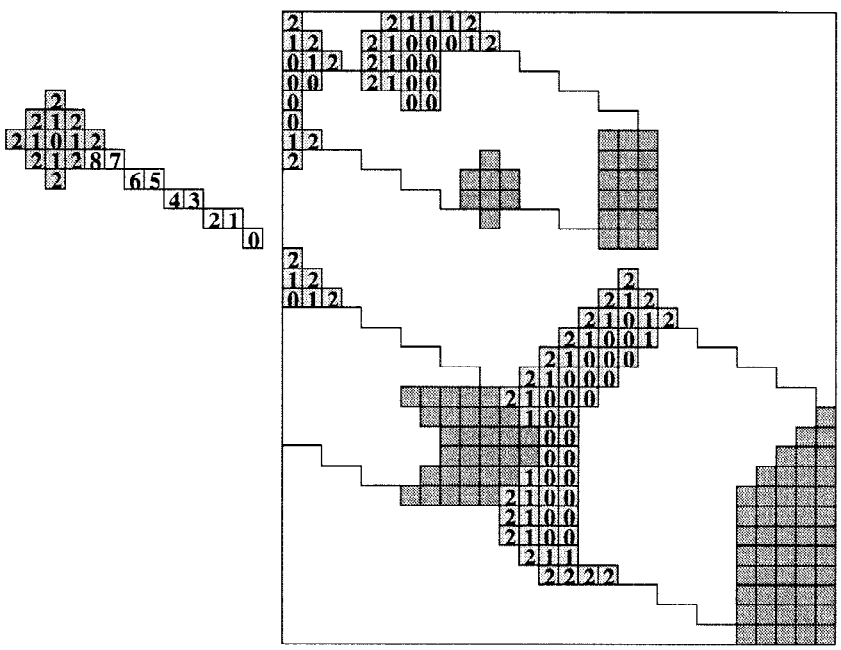


Fig. 20. $FP_\varepsilon(M, z)$: the shaded and numbered cells are those from D_ε that do not lie under E_ε. The original M is included for reference.

from which a rangefinder could return the given range vector, in a world characterized by the map. We provided formal definitions for feasible poses relative to single range probes and relative to sets of range probes, both with and without uncertainty in the range data. We gave exact computational-geometric algorithms to compute feasible poses. Finally, we defined feasible poses in a rasterized framework, and provided algorithms to compute feasible poses using *rasterized computational geometry*. One of our robots, LILY, uses a laser rangefinder and these rasterized algorithms to perform self localization by computation of feasible poses. In Section 3 we show examples of our localization system in operation.

3. Experiments and Results

3.1. *Research Program*. In this section we describe and present the results of experiments we performed to test the rasterized localization algorithm. We show maps that our robot, LILY (Figure 21) has made of two rooms within our building as well as the

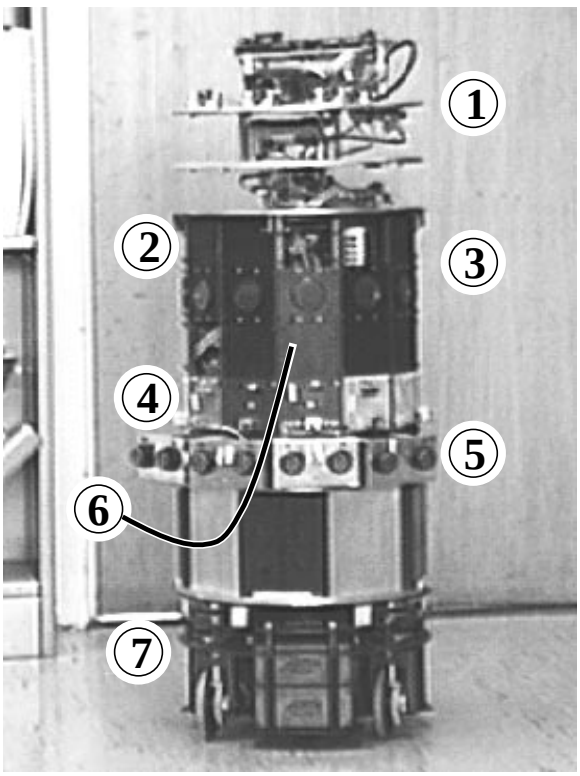


Fig. 21. LILY, one of the Cornell Mobile Robots: 1. The pan/tilt servo-able laser rangefinder. 2. The modular enclosure. 3. Sonar wide-angle range sensors. 4. Infrared directional modems and proximity detectors. 5. Contact sensors. 6. The modular, distributed, SCHEME-based robot controller. 7. Synchrodrive wheel-base (RWI, Inc.).

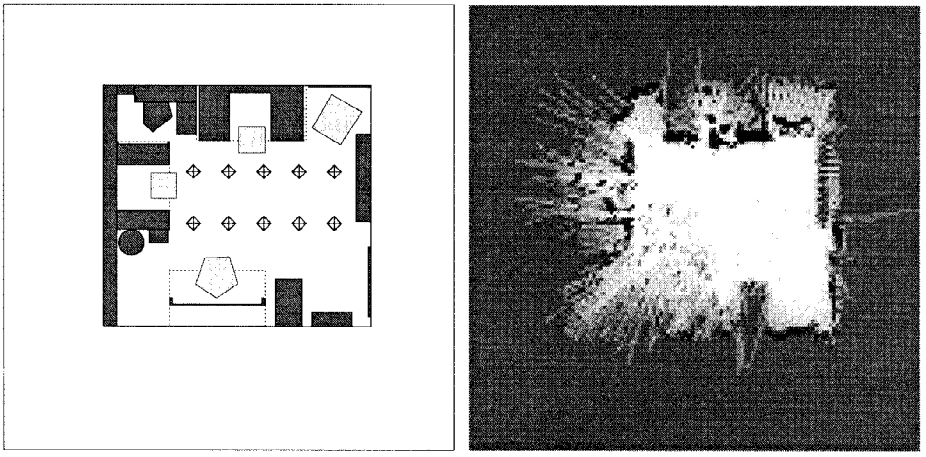


Fig. 22. (a) A hand-drawn map of Russell's office at Cornell. The small diamonds are the places where LILY sat while making the map shown in (b) At each location, she performed map updates based on 300 readings taken while she sat stationary and her pan-tilt head swept around the circle.

results of five different runs of the localization algorithm, wherein LILY used the maps she made, along with instantaneous range data, to localize herself within those rooms.

3.2. *Map-making and Localization Experiments.* Figures 22–31 show results of map-making and localization experiments performed using LILY. Figures 22–24 show the results of experiments run in a small (about 4 m $\times$ 4 m) office. Figures 25–31 show the results of experiments run in our laboratory (with dimensions roughly 7 m $\times$ 9 m). For these experiments, we specified a map size of 256 $\times$ 256. The scale factor between the world and the map was 50 mm $=$ 1 cell. This let us build maps of areas with size up to 12.8 m $\times$ 12.8 m.[5] In the maps generated by LILY, the white regions are those which are most certainly vacant, while the darkest regions are those which are most certainly the boundaries of obstacles. Grey regions are those for which either the robot was unable to obtain data (for example, the interiors of obstacles) or the region was ambiguous (e.g., occupied at some heights and vacant at others). LILY stores her map as an array of 8-bit values (0 to 255). Since we want a binary map for the localization algorithm, LILY converts the occupancy grid to a bitmap by applying a threshold at an occupancy probability of 50% $-$ ε (ε, here, just means "a small value"). In practice, LILY assigns a value of "occupied" in the binary map to any location in the statistical occupancy map holding a value greater than 127. Since 50% denotes "no information," this assigns a value of "occupied" in the bitmap to any cell for which we lack evidence that it is unoccupied. The same threshold was used for all experiments.

[5] For the office examples, we have cropped the map and localization grids to 128 $\times$ 128 so we can show the pertinent portions of the map more clearly.

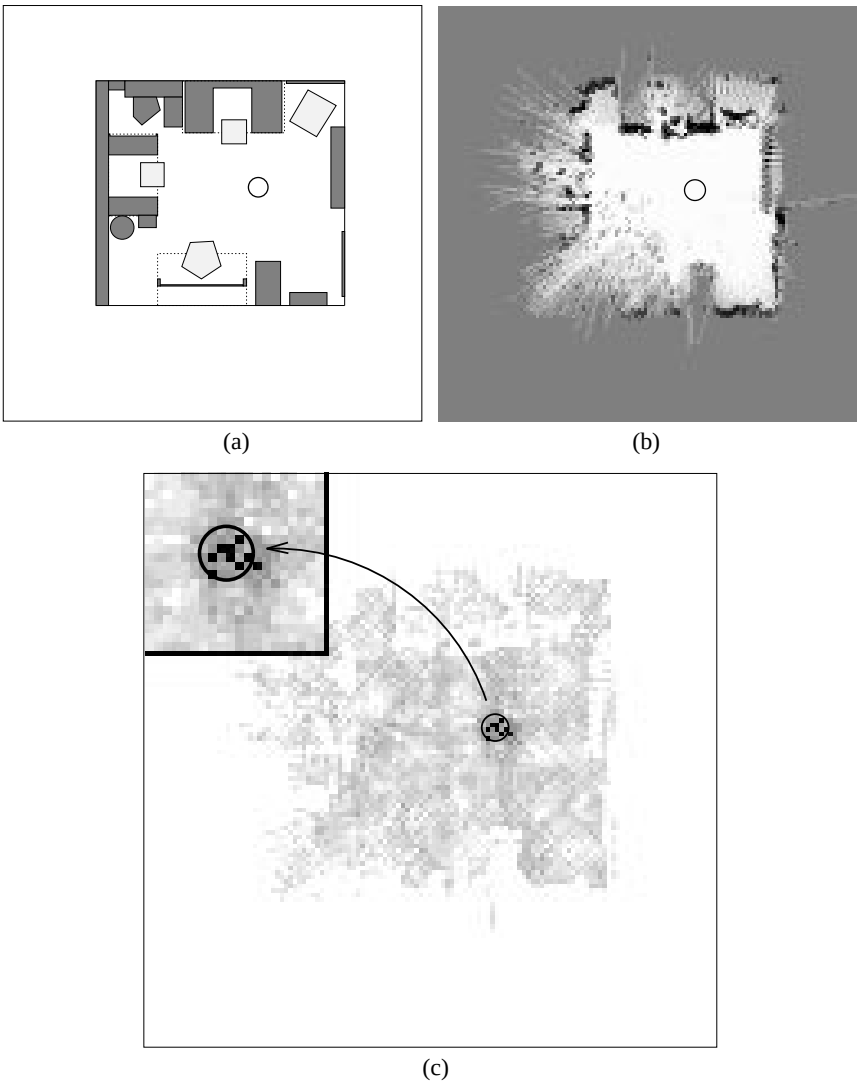
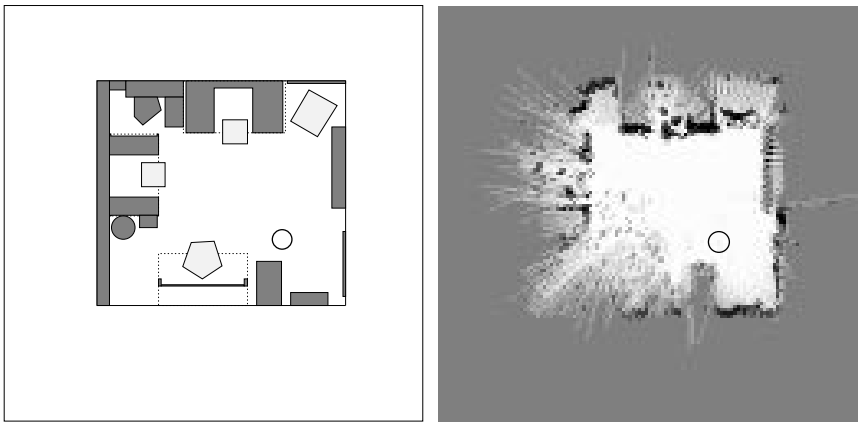
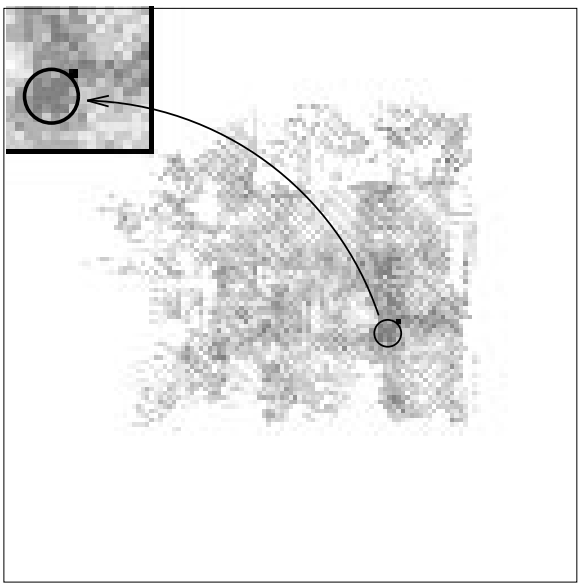


Fig. 23. The circles in these figures show the actual location and size of the robot for localization test 1 (important for judging the scale of the localization errors). The circles are overlaid on (a) the hand-drawn map, (b) LILY's map, and (c) the localization procedure output.

LILY builds these maps by the following process (see Figure 22(a)): The map is initialized with all elements at 50% probability of occupancy. LILY moves to a series of spatially separated positions, which can be a combination of hand-selected and automatically generated locations. At each position, LILY stops moving, and remains stationary while her laser rangefinder slowly spins around in a circle, taking 300 range readings. For each reading, she performs a statistical update to the map based on her



(a) (b)

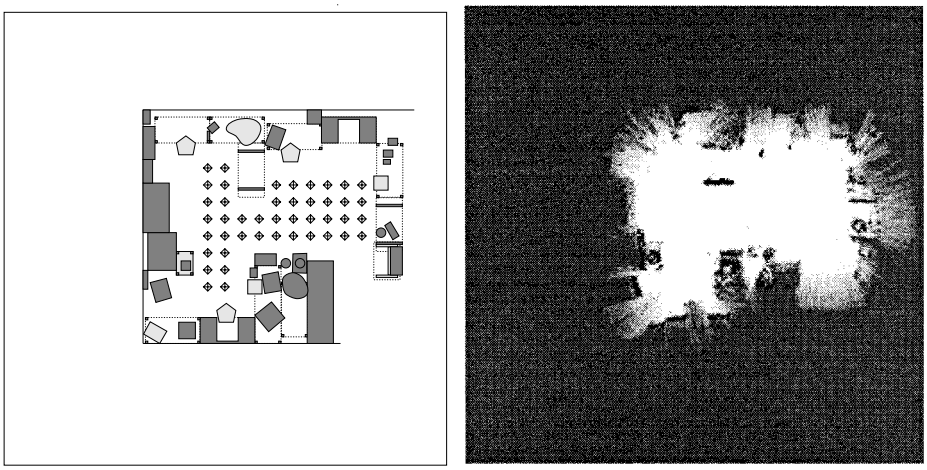


(c)

Fig. 24. The circles in these figures show the actual location and size of the robot for localization test 2. The circles are overlaid on (a) the hand-drawn map, (b) LILY's map, and (c) the localization procedure output.

perceived position and orientation, and the distance and direction values provided by the rangefinder.

The rule LILY used to update cells when it made the maps is an exponential-decay update rule: probability is raised by the rule $p_{\text{new}}(x) = 1 - (\alpha(1 - p_{\text{old}}(x)))$ and lowered by the rule $p_{\text{new}}(x) = \alpha p_{\text{old}}(x)$. In practice, we use distinct α_{raise} and α_{lower} values, for raising and lowering probabilities, respectively. The values we used in the examples shown in this section were 0.3 for α_{raise} and 0.15 for α_{lower} (the same values were used



(a) (b)

Fig. 25. (a) A hand-drawn map of our laboratory at Cornell. The small diamonds are the places where LILY sat while making the map shown in (b). At each location, she performed map updates based on 300 readings.

for all tests—no retuning was necessary). Also, since we implemented the occupancy grid as an array of bytes, it was adequate to implement the update rules as 256-entry lookup-tables (in effect, making each cell of the map a 256-state finite state machine), speeding the update procedure considerably.

In normal usage we slant the laser rangefinder toward the floor so that the laser beams hit the floor about two meters from the robot. The primary reason for this is to increase the reliability of the information we obtain from the sensor: If the rangefinder is level

(a) (b)

Fig. 26. The circles in these figures show the actual location and size of the robot for localization test 3. The circles are overlaid on (a) the hand-drawn map and (b) LILY's map.

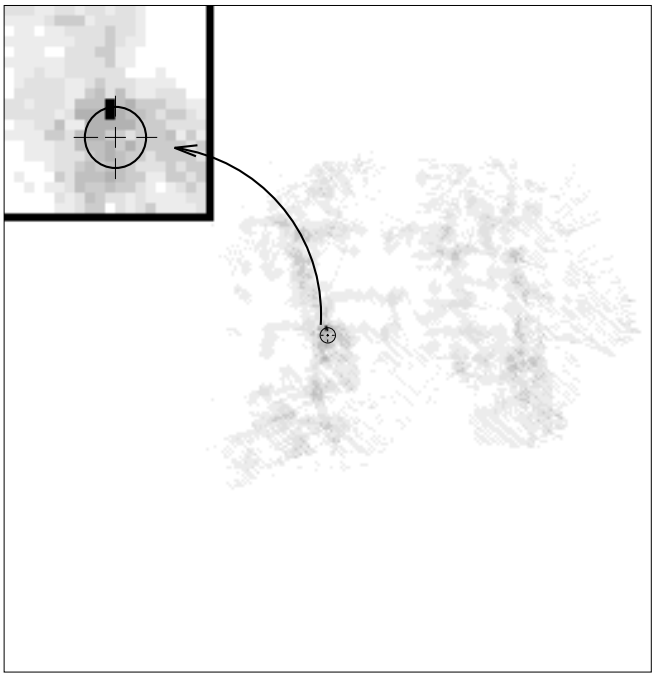


Fig. 27. The circle shows the actual location and size of the robot for localization test 3. It is overlaid on the localization procedure output.

(a) (b)

Fig. 28. The circles in these figures show the actual location and size of the robot for localization test 4. The circles are overlaid on (a) the hand-drawn map and (b) LILY's map.

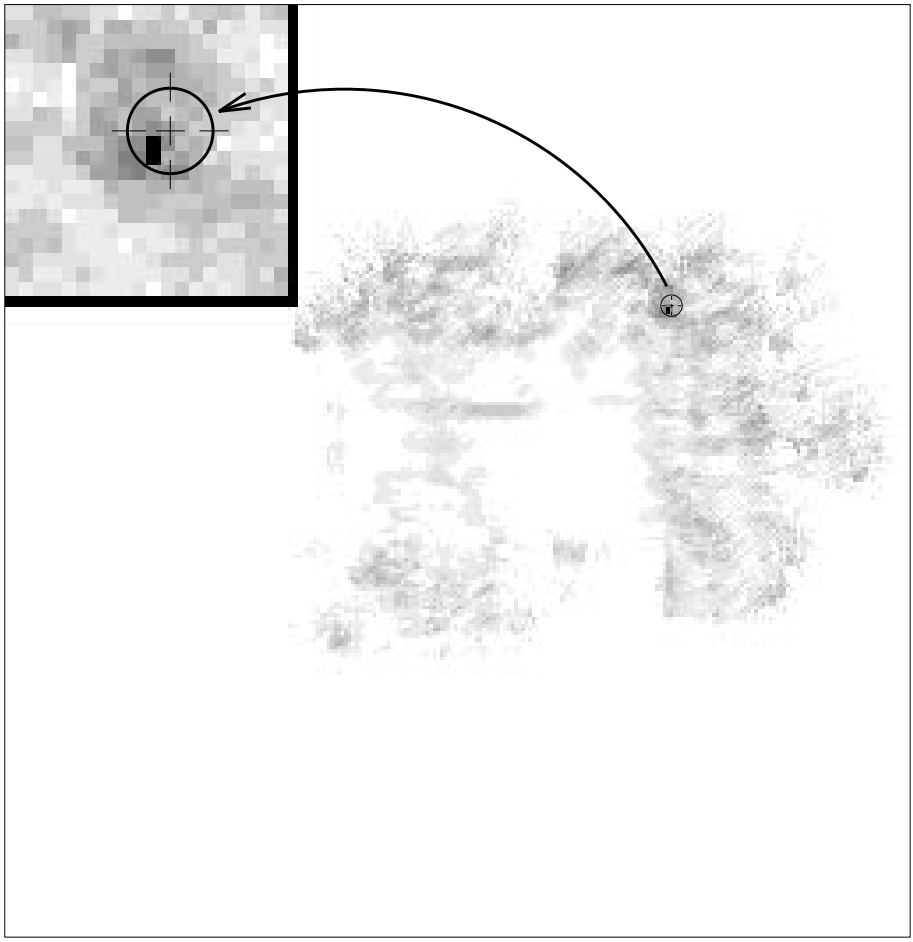


Fig. 29. The circle shows the actual location and size of the robot for localization test 4. It is overlaid on the localization procedure output.

to the floor, then its beams may not hit an object for a distance of several meters. If the objects hit are far away, the precision of the value returned is poor; furthermore, the odds of the sensor returning a completely false value are increased. In the map-making and localization experiments described in this section, we used the laser rangefinder in the down-slanted mode. Since the rangefinder was slanted toward the floor, we discarded those readings which were greater than 2.1 m, since those corresponded to the rangefinder sensing the floor, rather than an obstacle. For these experiments, we also discarded any readings smaller than the rangefinder's minimum range of 0.3 m.

In Figures 23 and 24 we show the results of two different runs of the localization procedure in the office environment. The (a) parts of these figures show the hand-drawn map of the office with a circle depicting the actual location of LILY; the circle has the same diameter as LILY (LILY has a cylindrical shape) drawn to scale. (Note that LILY never uses or sees the hand-drawn maps—these are only shown here for the reader's

(a) (b)

Fig. 30. The circles in these figures show the actual location and size of the robot for localization test 5. The circles are overlaid on (a) the hand-drawn map and (b) LILY's map.

convenience.) The (b) parts show the same circle overlaid on the map generated by LILY. The (c) parts show the output of the localization procedure run at the indicated location. In the localization output figures we have coded the display in the following way: cells which are *not* in $FP(M, z)$ for any of the chosen range probes are white. Cells which *are* in $FP(M, z)$ for one or more range probes are shown in shades of grey, such that darker cells are consistent with more probes. The darkest cells (the black ones)

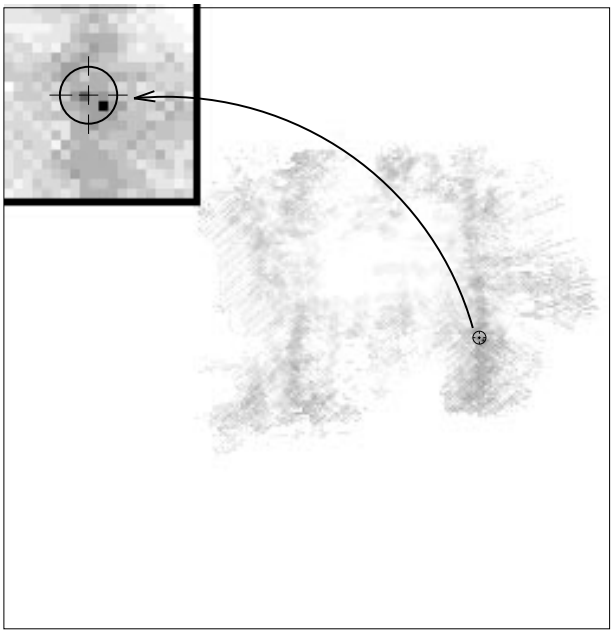


Fig. 31. The circle shows the actual location and size of the robot for localization test 5. It is overlaid on the localization procedure output.

are those which are consistent with a maximal number of probes. For these tests, we took probes at fixed intervals (one every 45°). In test 1 there were cells consistent with seven of the eight range probes. In test 2 there were cells consistent with six of the eight. Note that the darkest cells are all within or very near to the circle denoting LILY's position.

A further comment on interpreting these results: As pointed out, the circle is a to-scale representation of LILY; this means that a dark cell on the border of the circle represents a candidate feasible pose which is in error by approximately 100–200 mm, against the 6–13 m scale of the maps. This is due to a combination of measurement error in the actual range probes, and the cumulative effect of measurement error during the map-making process, exacerbated by the effect of varying cross-section objects in LILY's workspace.

Figures 26–31 show the results of three different runs of the localization procedure in the laboratory environment. The (a) and (b) parts of Figures 26, 28, and 30 are analogous to the (a) and (b) parts of Figures 23 and 24. Figures 27, 29, and 31 are analogous to the (c) parts of Figures 23 and 24. For these three runs of the localization procedure, the darkest pixels are, again, within or bordering on the circle denoting LILY's position for each run. In these cases, we took probes at 15° intervals. In test 3 seven probes were within acceptable range. In tests 4 and 5 ten and eleven probes were within range, respectively. For test 3 the maximal number of consistent range probes was six; this value was nine and eleven for tests 4 and 5.

3.3. *Future Experiments*. The test results we have obtained with our map-making and localization system seem to indicate that our approach to map-making and localization (combining a map-making system based on the work of Moravec and Elfes with a rasterized implementation of a computational geometry localization algorithm), using a point-and-shoot laser rangefinder, provides a promising avenue for mobile robot navigation. In this paper we can show only a few examples of this system in action; understandably, we have selected the ones which best illustrate our claims about the system. We have run the map-making and localization algorithms dozens of times on LILY, though, and have found them to be consistently reliable.[6]

Our navigation system is, clearly, still in the development stage, and there are many more experiments we would like to perform using it. We would like to extend the implementation of our navigation system to incorporate a number of other features that we have addressed in this paper. We have not yet been able to test many aspects of our navigation system, including

- **Three Dimensions**: We feel that the robot's maps would better reflect the environment if they incorporated height information. Approximating a three-dimensional world with a two-dimensional map reduces the accuracy of the map and the accuracy that one can expect from the localization algorithm.
- **Rotational Localization**: We have not yet tested our rotational localization strategies in concert with our translational localization algorithms.

[6] As the reader compares our work with competing algorithms, it is important to note that our localizations were performed on robot-made maps (not a priori hand-drawn maps), and real range data.

- **Simultaneous Map-Making and Localization**: For the examples shown in this section, it was the case that we made a map using dead reckoning, then used the localization algorithm to locate the robot after we moved the robot to an unspecified position. We would like to perform experiments in which the robot, for example, makes a map, or a portion thereof, then uses a combination of localization and dead reckoning to build an improved map; for these experiments, the robot would also determine for itself where it needed to explore to improve the quality of its map.

3.4. *Evaluation.* On the basis of the experiments outlined in this section, we feel that our localization algorithms performed very well. In addition to the results presented here, LILY has built well over a dozen maps, and performed about a hundred localization passes, with varying but generally very good results.

4. Conclusions and Further Research Directions

Closing Summary. We defined mobile robot localization as the process of finding *feasible poses*, locations that are consistent with the robot's internal map and with instantaneous range data. We presented algorithms to perform localization in $\mathbb{R}^2$ given a map and a set of range probes; the first of these were exact combinatorial algorithms which produced feasible pose sets from a polygonal map and a set of range probes. We then introduced the concept of *rasterized* algorithms, and presented rasterized algorithms to produce feasible poses when the map is a two-dimensional bitmap.

Finally, we presented the results of real localization experiments performed using LILY. We showed maps generated by LILY using her laser rangefinder and the results we obtained when we ran our localization algorithms on LILY using real of range data.

Further Research Directions. The work described in this paper suggests a number of interesting extensions at both theoretical and implementational levels. One set of extensions to the localization algorithms would be to extend them to $\mathbb{R}^3$ and $\mathbb{R}^3 \times S^1$. For the rasterized algorithms, this would primarily be an implementational extension, but for the exact algorithms there would be nontrivial complexity analysis required to obtain time bounds for localization in these higher dimensions. From a practical standpoint, it would be more interesting to determine whether $\mathbb{R}^3$ localization using $\mathbb{R}^3$ maps would overcome the difficulties encountered when we use $\mathbb{R}^2$ localization on an $\mathbb{R}^2$ map to find the robot's pose in an $\mathbb{R}^3$ world.

We have not yet explored in detail the use of history to improve the quality of localization estimates: We have only used previous pose estimates in the sense that we have assumed accurate orientation information. As a simple example, suppose we know from a previous execution of the localization routine that we are in, say, one of two or three small, disconnected regions. If we execute the localization routine again and determine that we are in one of a different set of two or three small, disconnected regions, and find that only one region in the new result is anywhere near any of the regions in the previous result, then, clearly, we know that we are most probably in that specific region.

Another promising direction would be to pursue active sensing issues in localization further, including (i) which range probes are most useful in localizing the robot and (ii) if

we cannot fully disambiguate the robot's current pose using instantaneous range data obtained from the robot's current pose, what is the best motion the robot can make to obtain data that will give it a more unique pose estimate? Work in this direction might be based on Kleinberg's on-line localization algorithm, presented in [Kle2].

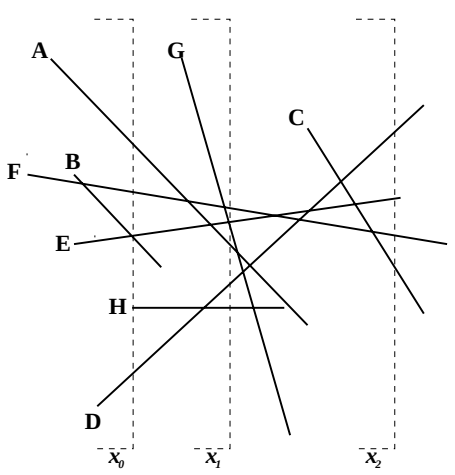
Conclusion. Much work has been done on mobile robot navigation problems. What we have done is to provide exact combinatorial algorithms with complexity analyses for the localization aspect of navigation and to provide rasterized versions of these algorithms which are suitable for execution upon a mobile robot. Both the exact algorithms and the rasterized algorithms are designed to handle sensor uncertainty robustly.

Acknowledgments. L. Paul Chew was very helpful in the development of the mobile robot localization algorithms described in this paper. The techniques we described here are a great deal simpler and more elegant than what we had before he volunteered his assistance, and we are most grateful for his assistance. Thanks also to the anonymous reviewer for helpful comments regarding Lemma 2.

The Cornell Mobile Robots, including LILY, are the result of many people's efforts. The list has grown too long to be presented fully, but we would like to mention some of those without whose help the experiments presented in this paper would not have been possible. First and foremost, we would like to thank James S. Jennings, who specified much of the architecture of our mobots and wrote the Cornell Generic Controller Monitor, which is the low-level operating system which runs on the mobots' low-level controllers. Jim also played a big role in specifying many of the mobots' sensor subsystems. Jonathan Rees implemented the on-robot SCHEME interpreter which makes it so easy to program the robots. Kevin Newman, Craig Becker, and Greg Whelan, among others, helped in the design, construction, and debugging of most of the subsystems. Mark Battisti and David Manzanares did the mechanical design and construction for several subsystems, including the pan/tilt head and chassis for the laser rangefinder and the supplemental battery chassis that made the longer experiments possible. We also wish to thank Grinnell More and Tyson Sawyer of Real World Interfaces, Inc., for the work they have done on the behalf of our mobile robotics program.

Appendix. The Plane Sweep. In this appendix, we review the family of algorithms known as "plane sweeps." The following discussion is a condensed treatment of material found in, among other places, [PS], [NP], [Don], and [Lat]. Suppose we have a set of n line segments in the plane, and we want to determine all of the intersections between them. The obvious way to do this is to go through all $(n)(n-1)/2$ pairs of segments, determine whether they intersect, and if so, where, and output all of the intersections. The cost of doing this is $O(n^2)$, as it takes constant time to process each pair. However, this cost is completely independent of the actual number of intersections; we would like an algorithm that is cheaper than $O(n^2)$ if there are significantly fewer than $O(n^2)$ intersections.

The plane-sweep segment-intersection algorithm meets that cost condition. The idea behind this algorithm is that we have a vertical line that we sweep across the plane,

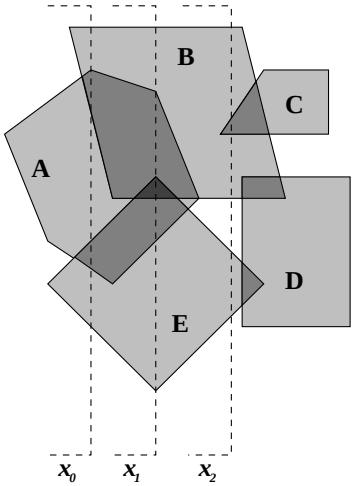


Sweep Line Position		
x_0	x_1	x_2
A	F	C
F	G/E	D
B/E	A	E
H	D	F
D	H	
Begin		
H		
End		
		E
Cross		
B,E	G,E	

Fig. 32. An example of the segment-intersecting plane sweep in operation.

stopping whenever we encounter a significant event. We keep two data structures, the sweep-line and the event queue, which contain information regarding the intersection of the sweep-line with objects in the plane, and upcoming events in the sweep, respectively. The event queue, E, contains scheduled future events, in this case, left and right endpoints of line segments and intersections of segments, supports the following operations: FIRST(E), INSERT(x, E), and MEMBER(x, E). By using a balanced-tree implementation, we can make these operations require at most log-time. The sweep-line, S, contains a description of the intersection of the sweep-line with the geometric structure being swept. We need S to support the operations INSERT(x, S), DELETE(x, S), and LEFT- and RIGHT-NEIGHBOR(x, S). In this case the main information contained in S is the vertical position of all segments that currently cross the sweep-line. Figure 32 shows a sample set of line segments and the position of the sweep-line at three of the events that occur during the sweep. The chart at the right shows the list of segments that the sweep-line encounters in top-to-bottom order for each event, as well as the event(s) that occur at that sweep position. Initially, we set the event queue to contain the x-coordinate of all segment endpoints, and set the sweep-line to be empty. At each event, we have to update both S and E. At any event, we delete that event from E and use the FIRST operator to determine the x-coordinate of the next event. At the left end of a segment ℓ, we (i) insert ℓ's y-coordinate and slope into S, and (ii) compute and insert into E the x-coordinate of the intersection(s) of ℓ with its neighbor segments in S. At the right end of ℓ, we delete ℓ from S. At the crossing of ℓ and another segment, q, we (i) delete ℓ and q from S, (ii) insert them into S again in the opposite order, and (iii) compute and insert into E the x-coordinate of any intersections between ℓ or q and their new neighbors.

The initialization cost of the algorithm is $O(n \log n)$: we must insert all of the left and right endpoints onto E. All of the event-driven updates take time $O(\log k)$ when there are k items in S or E. Since there are only n segments, the size of S will always be $O(n)$. E will likewise have $O(n)$ elements, since there are at most the n left endpoints,



Sweep Line Position					
x_0		x_1		x_2	
Polygon Transition/Coverage					
B+	0	B+	0	B+	0
A+	1	A+	1	C+	1
B-	2	E+	2	C-	2
E+	1	B-	3	B-	1
A-	2	A-	2	E+	0
E-	1	E-	1	E-	1
	0		0		0

Fig. 33. Computing polygon depth coverage with a plane sweep.

the n right endpoints, and a number of intersections that is no larger than the number of elements in S. Thus all events will take time $O(\log n)$. If there are c intersections overall, then the cost of running the plane sweep will be $O((n+c)\log n)$. If c is $\Omega(n^2/\log n)$, then the plane sweep will be at least as expensive as the brute-force approach, but in most cases arising in practice, the plane sweep will be less expensive.

Another major advantage of the plane-sweep approach is that we can do a lot more than just find intersections, without paying an asymptotic time penalty. Figure 33 shows a collection of overlapping polygons with the areas of overlap shaded proportionally to the depth of coverage. We can use a plane-sweep algorithm to compute all of the areas of overlap of a set of polygons, along with the depths of coverage. The output of this algorithm is known as the *arrangement* of the input polygons. For this sweep, each entry in the sweep-line structure contains an integer that is the depth of coverage at the associated segment of the sweep-line (0 for free-space, 1 when inside one polygon, 2 when in two, and so on) and also a pointer to the list of edges that describe the left-of-sweep portion of the polygonal region that that segment of the sweep-line passes through. When an intersection occurs, if it closes off a polygonal region, we output that region's boundary and depth. This does not increase the cost of the algorithm, since we can amortize the cost of outputting the region's boundary down to $O(1)$ time per vertex of that boundary. The chart on the right of Figure 33 shows the state of the sweep-line at three x-coordinates (not necessarily the x-coordinates of events).

We can compute more sophisticated coverage information using almost the same algorithm. The sweep algorithm can be used with various sorts of generalized polygons. Of particular interest to us here is the class of generalized polygons whose edges are line segments and circular arcs. The only fundamental change we must make to the algorithm is to insert any x-coordinate where a circular arc is tangent to the sweep-line; since there are only a constant number of these per arc (≤ 2), the performance analysis does not change.

We can also compute more specific arrangements. For example, in Figure 33, suppose that polygons **A**, **C**, and **D** are colored red, while polygons **B** and **E** are colored blue. We can compute both red coverage and blue coverage at once, in the same manner as we compute overall coverage, by keeping an array of color depths in each sweep-line entry. This can still be done with the same time bounds as before, if we use some care about how we track and output coverage information. The ability to compute multicolored arrangements in this fashion is a key part of the localization algorithms we will describe next. The plane sweep algorithm is a key subpart of many computational geometric algorithms. For other instances of applications using the plane sweep see, for example, [PS], [Don], and [Lat]).

References

[AH] S. Atiya and G. Hager. Real-time vision-based robot localization. In *Proc*. 1991 *IEEE International Conference on Robotics and Automation*, pages 639–644, Sacramento, CA, 1991.

[AHU] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA, 1983.

[AST] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. In *Proc. Third ACM–SIAM Symposium on Discrete Algorithms*, pages 72–82, 1992.

[BL] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, **10**(6):628–649, 1991.

[BR] J. R. Beveridge and E. M. Riseman. Hallway Navigation in Perspective. Technical report, Computer and Information Sciences Department, University of Massachusetts, Amherst, MA, June 1991.

[Bro] R. G. Brown. Algorithms for Mobile Robot Localization and Building Flexible, Robust, Easy to Use Mobile Robots. Ph.D. thesis, Cornell University, Ithaca, NY, May 1995.

[CC] F. Chenavier and J. L. Crowley. Position estimation for a mobile robot using vision and odometry. In *Proc*. 1992 *IEEE International Conference on Robotics and Automation*, pages 2588–2593, Nice, 1992.

[CGH$^+$] L. P. Chew, M. T. Goodrich, D. P. Huttenlocher, K. Kedem, J. M. Kleinberg, and D. Kravets. Geometric pattern matching under Euclidean motion. In *Proc*. *Fifth Canadian Conference on Computational Geometry*, pages 151–156, Waterloo, Ontario, August 1993.

[CK] L. P. Chew and K. Kedem. Improvements on approximate pattern matching problems. In O. Nurmi and E. Ukkonen, editors, *Proc*. *Third Scandinavian Workshop on Algorithm Theory*. Lecture Notes in Computer Science, Volume 621. Springer-Verlag, Berlin, 1992.

[DKM] B. R. Donald, D. Kapur, and J. Mundy. *Symbolic and Numerical Computation for Artificial Intelligence*. Academic Press, Harcourt Jovanovich, New York, 1992.

[Don] B. Donald. *Error Detection and Recovery in Robotics*, Lecture Notes in Computer Science, volume 336. Springer-Verlag, New York, 1989.

[Dru] Drumheller. Mobile robot localization using sonar. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **9**(2), 1987.

[Elf] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, **3**(3), 1987.

[GLP] W. E. L. Grimson and T. Lozano-Pérez. Recognition and localization of overlapping parts from sparse data in two and three dimensions. In *Proc*. 1985 *IEEE International Conference on Robotics and Automation*, pages 61–66, St. Louis, MO, 1985.

[GMR] L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem in two dimensions. In *Proc*. *Symposium on Discrete Algorithms*, pages 259–268, 1991.

[GSO] J. Gonzalez, A. Stentz, and A. Ollero. An iconic position estimator for a 2d laser rangefinder. In *Proc*. 1992 *IEEE International Conference on Robotics and Automation*, pages 2646–2651, Nice, 1992.

[HK1] D. P. Huttenlocher and K. Kedem. Efficiently computing the hausdorff distance for point sets under translation. In *Proc. Sixth ACM Symposium on Computational Geometry*, pages 340–349, 1990.

[HK2] D. P. Huttenlocher and K. Kedem. Distance metrics for comparing shapes in the plane. In *Symbolic and Numerical Computation for Artificial Intelligence*, pages 201–219. Academic Press, New York, 1993.

[HMB] A. A. Holenstein, M. A. Müller, and E. Badreddin. Mobile robot localization in a structured environment cluttered with obstacles. In *Proc*. 1992 *IEEE International Conference on Robotics and Automation*, pages 2576–2581, Nice, 1992.

[Jaz] A. H. Jazwinski. *Stochastic Processes and Filtering Theory*. Academic Press, New York, 1970.

[KK] I. S. Kweon and T. Kanade. Extracting topographic features for outdoor mobile robotics. In *Proc*. 1991 *IEEE International Conference on Robotics and Automation*, pages 1992–1997, Sacramento, CA, 1991.

[Kle1] L. Kleeman. Optimal estimation of position and heading for mobile robots using ultrasonic beacons and dead-reckoning. In *Proc*. 1992 *IEEE International Conference on Robotics and Automation*, pages 2582–2587, Nice, 1992.

[Kle2] J. Kleinberg. The localization problem for mobile robots. In *Proc*. 35*th IEEE Symposium on Foundations of Computer Science*, 1994.

[Lat] J.-C. Latombe. *Robot Motion planning*. Kluwer, Dordrecht, 1992.

[LDWC] J. Leonard, H. Durrant-Whyte, and I. J. Cox. Dynamic map building for an autonomous mobile robot. In *Proc*. 1990 *IEEE/RSJ International Conference on Intelligent Robot Systems*, 1990.

[LP] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, **32**(2):108–120, 1983. Also MIT A.I. Memo 605, Dec. 1982.

[LRDG] J. Lengyel, M. Reichert, B. Donald, and D. Greenberg. Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware. Department of Computer Science Technical Report TR 90-1122, Department of Computer Science, Cornell University, Ithaca, NY, May 1990.

[Mar] D. Marr. *Vision*. Freeman, New York, 1982.

[MDW] J. M. Manyika and H. F. Durrant-Whyte. A tracking sonar sensor for vehicle guidance. In *Proc*. 1993 *IEEE International Conference on Robotics and Automation*, volume 2, pages 424–429, Atlanta, Ga, 1993.

[ME] H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc*. 1985 *IEEE International Conference on Robotics and Automation*, pages 116–121, St. Louis, MO, 1985.

[Meg] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the Association for Computing Machinery*, **30**:852–865, 1983.

[MH] D. Marr and E. Hildreth. Theory of edge detection. Artificial Intelligence Laboratory Memo 518, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, April 1979.

[Mor] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. In A. Casals, editor, *Sensor Devices and Systems for Robotics*. Springer-Verlag, New York, 1989.

[MR] C. D. McGillem and T. S. Rappaport. Infra-red location system for navigation of autonomous vehicles. In *Proc*. 1988 *IEEE International Conference on Robotics and Automation*, pages 1236–1238, Philadelphia, PA, 1988.

[NP] J. Neivergelt and F. P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Communications of the ACM*, **25**(10):739–747, 1982.

[OS] A. V. Oppenheim and R. W. Schafer. *Discrete Time Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[OW] A. V. Oppenheim and A. S. Willsky. *Signals and Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[PS] F. P. Preparata and M. Shamos. *Computational Geometry*: *An Introduction*. Springer-Verlag, New York, 1985.

[RG] S. Ratering and M. Gini. Robot navigation in a known environment with unknown moving obstacles. In *Proc*. 1993 *IEEE International Conference on Robotics and Automation*, volume 1, pages 25–30, Atlanta, Ga, 1993.

[Ruc] W. J. Rucklidge. Lower bounds for the complexity of the Hausdorff distance. In *Proc. Fifth Canadian Conference on Computational Geometry*, pages 145–150, Waterloo, Ontario, August 1993.

[RWA+] Y. Roth, A. S. Wu, R. H. Arpaci, T. Weymouth, and R. Jain. Model-driven pose correction. In *Proc*. 1992 *IEEE International Conference on Robotics and Automation*, pages 2625–2630, Nice, 1992.

[SHD] T. M. Silberberg, D. A. Harwood, and L. S. Davis. Object recognition using oriented model points. *Computer Vision, Graphics and Image Processing*, **35**:47–71, 1986.

[TL] H. Takeda and J. C. Latombe. Planning the Motions of a Mobile Robot in a Sensory Uncertainty Field. Stanford CS Technical Report STAN-C-92-1424, Department of Computer Science, Stanford University, Stanford, CA, April 1992.