Navigation Strategies for Exploring Indoor Environments

Héctor H. González-Baños[(1)] Jean-Claude Latombe[(2)]

(1) Honda R&D, Americas; 800 California St. Suite 300; Mountain View, CA 94041

(2) Department of Computer Science; Stanford University; Stanford, CA 94305

e-mail:{hhg,latombe}@robotics.stanford.edu

Abstract

This article presents a system in which a mobile robot equipped with a range sensor efficiently builds polygonal layouts of indoor environments as it navigates. It is assumed that no prior information about the environment is available. The model (layout) is constructed concurrently as the environment is explored. At each step, the robot decides where to move next (sensor placement problem) based on the current partially-built model. This decision is made by an algorithm that guides the robot through a sequence of "good" views, with "good" referring to the expected amount and quality of the information that will be revealed at each new view. This is the so-called *next-best view* (NBV) problem, which also appears in domains such as Computer Vision and Computer Graphics. In mobile robotics, however, this problem is complicated by a number of implementation issues, two of which are particularly crucial. One is the safe navigation of the robot, despite the fact that its knowledge about the environment is incomplete. The other is the need to precisely align successive views, despite positioning uncertainty inherent to mobile robots. To address these issues, this article introduces the concept of a *safe region* – the largest region guaranteed to be free of obstacles given the sensor readings made so far. It proposes a NBV algorithm based on this concept, which takes sensor limitations (range and incidence) into account.

1 Introduction

Automatic model building is a fundamental task in mobile robotics [2, 24]. The basic problem is easy to formulate: After being deployed into an unknown environment, a robot, or a team of robots, must perform sensing operations at multiple locations and integrate the acquired data into a representation of the environment. Despite this simple formulation, the problem is difficult to solve in practice. First, there is the problem of choosing an adequate representation of the environment — e.g., topological maps [3], polygonal layouts [2], occupancy grids [7], 3-D models [23], or feature-based maps [11]. Second, the representation must be extracted from imperfect sensor readings — e.g., depth readings from range-sensors may fluctuate due to changes in surface textures [5], different sets of 3-D scans must be zippered (stitched) [25], and captured images must be aligned and registered [21, 15]. Finally, if the system is truly automatic, the robot must decide on its own the necessary motions to construct the model [9, 10].

Past research in model construction has mainly focused on developing techniques for extracting relevant features (e.g., edges, corners) from raw sensor data, and on integrating these into a single and consistent model. There is also prior research on the computation of the next-best view, a problem that has attracted considerable attention (e.g., see [1, 4, 15, 21, 26]). Unfortunately, most of the proposed NBV techniques are unsuitable for mobile robotics for two reasons. First, the robot must navigate without colliding with uncharted obstacles (safe navigation). Second, it must be able to re-localize itself with respect to a partially-built map (image registration).

Typical NBV techniques do not address safe navigation constraints because they are designed for systems that build a model of a relatively small object using a range sensor moving around the specimen. Collisions, however, are not a major issue for sensors that operate outside the convex hull of the scene of interest. In mobile robotics, by contrast, the sensor navigates *within* the scene's convex hull. Therefore, safe navigation considerations must be taken into account when computing the next-best view for a map-building robot.

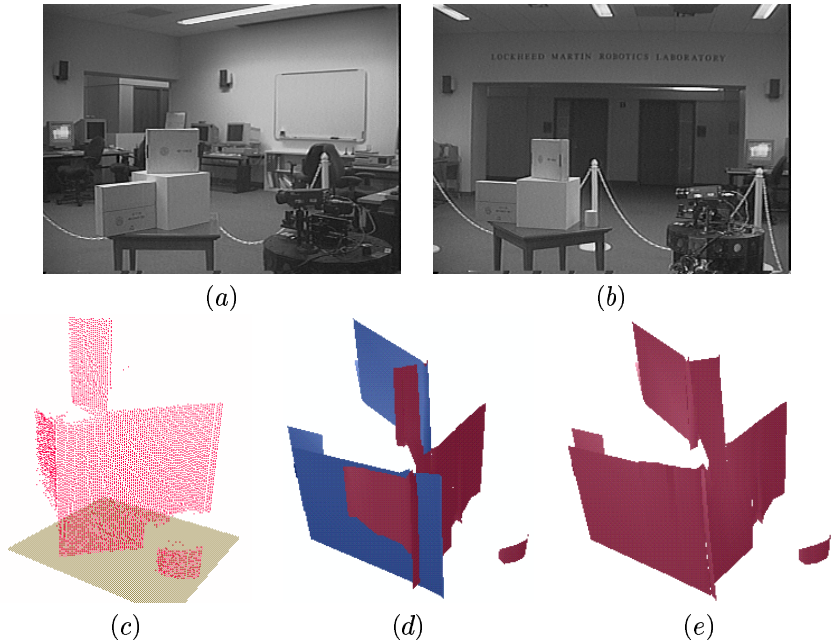


Figure 1: Image registration problem: A robot equipped with a laser range-finder scans a scene from locations (a) and (b); each raw image consists of a set of points in space (c); if the views are superimposed relying exclusively upon the data provided by the robot's encoders then a mismatch occurs (d); an algorithm that aligns surfaces constructed from the data sets is able to correct the mismatch and merge both images (e).

Moreover, a useful NBV technique must also consider image-alignment restrictions [20], a problem that is particularly fastidious in a mobile robot because of wheel slippage (see Figure 1). Although image alignment is a key step in the construction of visual models, it is also the classic robot localization problem seen from a different angle. In fact, for a map-building robot these two problems are equivalent and are referred as the *simultaneous localization and map building* (SLAM) problem [18, 13, 6]. Many image-alignment techniques can be found in the literature, but all of them require a minimum overlap between each new image and portions of the environment seen by the robot at previous sensing locations [18]. Therefore, a good NBV technique must guarantee that such overlap between images occurs.

In this article, we describe how the safe navigation and alignment/localization problems can be simultaneously addressed by using *safe regions*. These are the largest regions guaranteed to be safe given the sensor readings obtained so far. By using safe regions, it is possible to iteratively build a map by executing union operations over successive views (map building), and use this same map for motion planning (safe navigation). Additionally, safe regions can be used to anticipate the overlap between future views and the current partially-built map (image alignment), and to compute locations that could potentially see large unexplored areas (next-best view). The concept of safe regions offers a single framework to address these seemingly unrelated issues raised by the computation of the next-best view.

This article is divided in two parts. Part I introduces the formal definition of a safe region (Section 2), and the complexity of computing this region from sensor data (Section 3). The general form of our NBV algorithm is described in Section 4.

In Part II, we describe the implementation of an actual map-building robotic system. In order to actually build a layout of a building, in addition to the computation of the next-best view other operations must also take place. These operations include polyline generation from range data, and model alignment and merging (Section 6). The specific NBV algorithm embedded in our prototype (Section 7) is also described in the second half of this article. The system architecture and experiments are described in Section 8.

Finally, in Section 9 we describe the current major limitations of our work, as well as some important extensions to be investigated in future research.

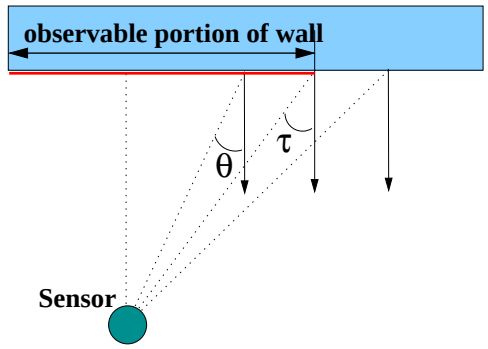


Figure 2: Incidence constraint: Wall sections are seen only if $\mid \theta \mid \leq \tau$.

Part I

In this first part of our article we will introduce the concept of safe region. A key result is Theorem 3.1, which states that a local safe region is no more complex than the visibility region computed under classic unrestricted visibility. The proof of this theorem also provides a constructive way of computing safe regions.

Safe regions are then used to iteratively construct a map. This approach leads very naturally to a general and flexible NBV algorithm. This algorithm is the central component of the system described in Part II.

2 Definition of Safe Regions

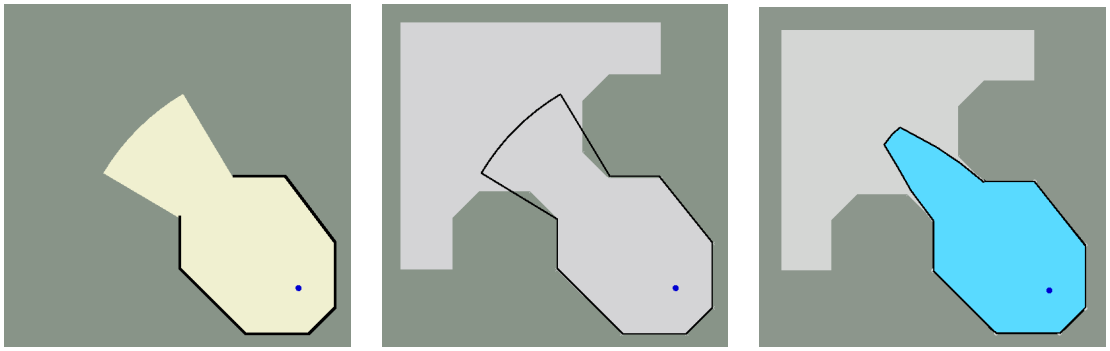
Suppose that the robot is equipped with a polar range sensor measuring the distance from the sensor's center to objects lying in a horizontal plane located at height h above the floor. Because all visual sensors are limited in range, we assume that objects can only be detected within a distance r_{max}. In addition, most range-finders cannot reliably detect surfaces oriented at grazing angles with respect to the sensor (see Figure 2). Hence, we also assume that surface points that do not satisfy the sensor's incidence constraint cannot be reliably detected by the sensor. Formally, our visibility model is the following:

Definition 2.1 (Visibility under Incidence and Range Constraints) *Let the open subset $\mathcal{W} \subset \Re^2$ describe the workspace layout. Let $\partial\mathcal{W}$ be the boundary of $\mathcal{W}$. A point $\boldsymbol{w} \in \partial\mathcal{W}$ is* visible *from a point $\boldsymbol{q} \in \mathcal{W}$ if the following conditions are true:*

1. Line of sight constraint*: The open line segment $S(\boldsymbol{w}, \boldsymbol{q})$ joining $\boldsymbol{q}$ and $\boldsymbol{w}$ does not intersect $\partial\mathcal{W}$.*
2. Range constraint*: $d(\boldsymbol{q}, \boldsymbol{w}) \leq r_{max}$, where $d(\boldsymbol{q}, \boldsymbol{w})$ is the Euclidean distance between $\boldsymbol{q}$ and $\boldsymbol{w}$, and $r_{max} > 0$ is an input constant.*
3. Incidence constraint*: $\angle(\boldsymbol{n}, \boldsymbol{v}) \leq \tau$, where $\boldsymbol{n}$ is a vector perpendicular to $\partial\mathcal{W}$ at $\boldsymbol{w}$, $\boldsymbol{v}$ a vector oriented from $\boldsymbol{w}$ to $\boldsymbol{q}$, and $\tau \in [0, \pi/2]$ is an input constant* (see Figure 2).

Without any loss of generality, we assume that the sensor is located at the origin of the coordinate system (the workspace can always be re-mapped to a reference frame centered on the sensor). Let $\mathcal{W} \subset \Re^2$ describe the workspace, and let $\partial\mathcal{W}$ be the boundary of $\mathcal{W}$. The sensor's output is assumed to be as follows:

Definition 2.2 (Range Sensor Output) *The output of a range sensor is an ordered list Π, representing the sections of $\partial\mathcal{W}$ visible from the origin under Definition 2.1. Every $r(\theta; a, b) \in \Pi$ is a polar function describing a section of $\partial\mathcal{W}$, and such function is continuous $\forall\ \theta \in (a, b)$ and undefined elsewhere. Π contains at most one function defined for any $\theta \in (-\pi, \pi]$ (i.e., no two functions overlap), and the list is ordered counter-clockwise.*



(a) (b) (c)

Figure 3: Effect of incidence on safe regions.

Given an observation Π made by the robot at a location q, we define the *local safe region* $s_l(q)$ as the largest region guaranteed to be free of obstacles. While range restrictions have an obvious impact on $s_l(q)$, the effect of incidence is more subtle. In Figure 3(a), a sensor detects the surface contour shown in black. A naive approach may join the detected surfaces to the perimeter limit of the sensor, and consider this region free from obstacles (the region is shown in light color). Because the sensor is unable to detect surfaces oriented at grazing angles, this region may be not be safe, as shown in (b). A true safe region is shown in (c), for an incidence constraint of $\tau = 70$ deg.

3 Computational Complexity of Safe Regions

Let ∂s_l be the boundary of the region s_l. ∂s_l is composed by solid and free curves. A *solid curve* represents an observed section of $\partial\mathcal{W}$, and is contained in the list Π.

Given two solid curves $\{r_1(\theta; a_1, b_1), r_2(\theta; a_2, b_2)\} \subseteq \Pi$, r_2 is said to *succeed* r_1 if no other element in Π is defined in the interval $[b_1, a_2]$. A curve $f(\theta; b_1, a_2)$ joining a pair (r_1, r_2) of successive sections is called a *free curve* if: (1) no undetected obstacle is contained in the polar region $b_1 < \theta < a_2$ bounded by f; and (2) this region is the largest possible.

In order to compute the local safe region at q we need to compute the free curves that join each successive pair in Π in order to bound the region $s_l(q)$. It turns out that the complexity of f is $\mathrm{O}(1)$. In fact, a free curve f can be described using no more than 3 function primitives:

Theorem 3.1 (Free Curves) *Suppose $r_2(\theta; a_2, b_2)$ succeeds $r_1(\theta; a_1, b_1)$ in the output list Π of a sensor operating under Definition 2.2 and located at the origin. If $\partial\mathcal{W}$ is continuously differentiable, then the free curve $f(\theta; b_1, a_2)$ connecting r_1 to r_2 consists of at most three pieces. Each piece is either a line segment, a circular arc, or a section of a logarithmic spiral of the form $r = r_o \exp(\pm\lambda\theta)$ (where $\lambda = \tan\tau$ and r_o is a constant).*

Note that the theorem also applies when the sensor is located at an arbitrary position q: Given the observation $\Pi(q)$, re-map the data to a reference frame centered at q and call this Π'. Once the free curves are computed, apply the inverse transformation to express the results in global coordinates.

The proof of Theorem 3.1 is long and tedious, and is therefore included in the appendix. But the main consequence of the theorem is that a free curve f joining r_1 with r_2 is a function of only the sensor constraints and the endpoints of r_1 and r_2. That is, f is *independent* of the complexity of the workspace $\mathcal{W}$ and can be computed in constant time. Indeed, if $\Pi(q)$ contains m solid curves, then $\partial s_l(q)$ contains no more than $3m$ additional function primitives. Because the complexity of $\Pi(q)$ cannot be less than m, the complexity of $s_l(q)$ is within a constant proportionality factor from the complexity of $\Pi(q)$.

The proof is based on a continuity argument. Although the shape of the workspace may be arbitrary, $\mathcal{W}$ represents a physical space containing physical objects. Its boundary $\partial\mathcal{W}$ consists of a set of Jordan

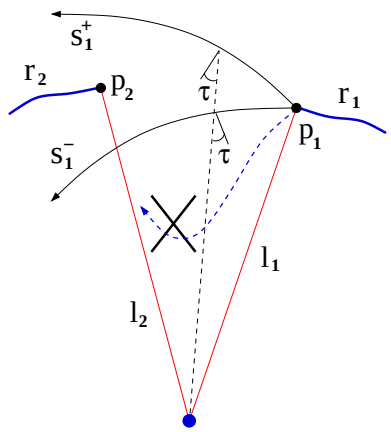


Figure 4: An example of a free curve construction.

curves (i.e., closed loops in the plane which do not intersect themselves), and each Jordan curve is in itself a continuous curve. If additionally we enforce that the curve is differentiable, then we can show that the unobserved section of ∂W lying between a successive pair of solid curves cannot be arbitrarily close to the sensor. To illustrate this point, we sketch here one case of the proof.

Let $r_1(\theta; a_1, b_1)$ and $r_2(\theta; a_2, b_2)$ be a successive pair of solid curves, and suppose that the incidence constraint was exceeded immediately after $\theta = b_1$ (Figure 4). Therefore, the normal to $\partial\mathcal{W}$ immediately after r_1 is oriented at a grazing angle with respect to the sensor. What can we say about the unobserved section of ∂W lying between r_1 and r_2?

Define the points $p_1 = (\rho_1, b_1)$ and $p_2 = (\rho_2, a_2)$, where $\rho_1 = r_1(b_1)$ and $\rho_2 = r_2(a_2)$ (see Figure 4). Suppose that the boundary of $\mathcal{W}$ continues after p_1 with its surface normal constantly oriented at *exactly* an angle τ with respect to the sensor's line-of-sight. This curve in polar coordinates has the form $r = r_o \exp[\pm\lambda(\theta - \theta_o)]$, where $r_o = \rho_1$ and $\theta_o = b_1$. This last equation defines two spirals: a spiral s_1^+ growing counter-clockwise from $p_1 = (\rho_1, b_1)$ (or shrinking clockwise), and a second spiral s_1^- shrinking counter-clockwise from p_1 (or growing clockwise). So far, we may conclude that $\partial\mathcal{W}$ must continue from p_1 in the counter-clockwise direction either "above" s_1^+ or "below" s_1^-; otherwise, the incidence constraint would not have been violated.

But $\partial\mathcal{W}$ cannot continue below s_1^- for the example shown in Figure 4. To show this, suppose $\partial\mathcal{W}$ continues below s_1^-, which implies that $\partial\mathcal{W}$ bends toward the sensor immediately after r_1. We know that $\partial\mathcal{W}$ does not cross the origin, else nothing will be visible under Definition 2.1 and Π would be empty. Thus, $\partial\mathcal{W}$ has to bend outwards; and it has to do so before cutting the ray l_2 joining the origin with p_2, otherwise r_2 will be occluded. Since $\partial\mathcal{W}$ is differentiable, there must be a point p where the normal to $\partial\mathcal{W}$ points towards the origin. Therefore, the vicinity of p is a visible portion of $\partial\mathcal{W}$. But this violates our assumption that r_2 succeeds r_1, so $\partial\mathcal{W}$ must continue above s_1^+.

Following a similar reasoning, it is possible to account for every possible combination of events, depending on whether the sensor line-of-sight was occluded, the range constraint was exceeded, or the incidence constraint was exceeded. In the appendix, all cases are described in detail.

3.1 Safe Region Properties

The region $s_l(q)$ is topologically equivalent to the classic visibility region. Indeed, when the sensor restrictions in Definition 2.1 are relaxed, the safe region becomes the visibility region. Therefore, several properties and algorithms that apply to visibility regions also apply to safe regions. For example, the segment connecting q with p is entirely contained in $s_l(q)$ for any $p \in s_l(q)$. That is, $s_l(q)$ is a member of the class of *star regions* — regions that are entirely observable from at least a single interior point called the region's kernel.

Likewise, a local safe region is a simply connected region — all paths joining a pair of points p_1 and p_2 inside $s_l(q)$ are *homotopic*. Two paths γ and γ' with endpoints fixed are homotopic if γ can be continuously

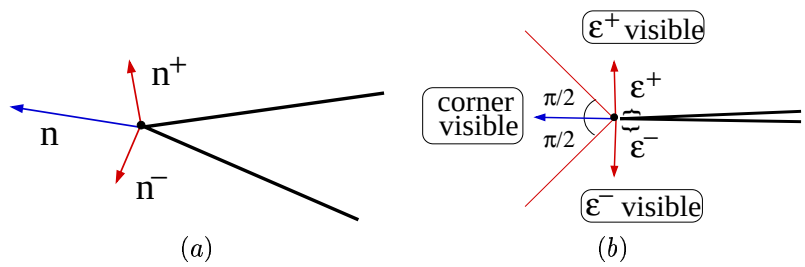


Figure 5: Dealing with corners: (a) the normal to $\partial\mathcal{W}$ at a corner is generalized as the average of $\boldsymbol{n}^+$ and $\boldsymbol{n}^-$; (b) if we assume that a corner has "thickness", and is therefore detectable by the sensor, then any wedge-shaped object is visible if $\tau \geq 45$ deg.

deformed into γ' by applying a continuous mapping (or vice-versa). This an important property, because in order to find an optimal path γ^* between p_1 and p_2 given some optimization criterion, it is possible to apply simple descent methods over the parameterization of γ in order to compute or approximate γ^*. In other words, a solution to a given path planning problem for fixed endpoints can be deformed into the solution of a harder problem for the same endpoints.[1]

For polygonal environments composed of n edges, $\partial s_l(p)$ contains no more than $\mathrm{O}(n)$ sections, which is the same upper bound as for visibility polygons. In fact, we can compute a tighter bound. Let $\mathcal{V}(q)$ be the visibility polygon computed at q. Let m be the number of visible (solid) edges in $\mathcal{V}(q)$. In the worst case, each visible edge in $\mathcal{V}(q)$ is only partially visible under Definition 2.1. Thus, the boundary of $s_l(q)$ is composed of m (possibly shorter) solid edges, plus at most m free curves joining each successive pair of solid edges. Since each free curve is consists of at most 3 sections, then $\partial s_l(q)$ is composed of no more than $4m$ sections, each of which can be either a segment, a circular arc or a section of a logarithmic spiral.

Finally, one must not forget that $s_l(q)$ is guaranteed to be free of obstacles. Because the workspace is not known in advance, it is very important to ensure that the robot will not collide with unseen obstacles before these are detected. Region $s_l(q)$ characterizes this obstacle-free region. The robot will not be in collision as long as its motions are restricted to $s_l(q)$ between successive sensing operations.

3.2 Corners

Corners pose a problem even under idealized conditions. Suppose a robot is surrounded by one or several wedge-shaped walls oriented toward the sensor. The sensor is then unable to see any of these wedges, and the safe region is empty. This is not a failure of our mathematical analysis, but a physical limitation of the sensor. This limitation is ignored by Definition 2.1, along with several others (e.g., that some surfaces could be perfectly transparent or completely reflective). We can only assume that the angle between any pair of incident walls is large enough such that at least one section at either side of the corner is visible to the sensor. Or that the corner itself is not sharp enough to remain undetected by the sensor (i.e., the corner has "thickness").

Under the above assumptions, we generalize the concept of a surface normal to include corners. The normal $\boldsymbol{n}$ to $\partial\mathcal{W}$ at a corner is the average of $\boldsymbol{n}^+$ and $\boldsymbol{n}^-$, where $\boldsymbol{n}^+$ and $\boldsymbol{n}^-$ are the normals to $\partial\mathcal{W}$ immediately after and before the corner (Figure 5(a)). The corner is visible if the conditions of Definition 2.1 are satisfied for this generalized $\boldsymbol{n}$. That is, a corner behaves like any other point in $\partial\mathcal{W}$, as long as our hypotheses for $\partial\mathcal{W}$ hold true.

The system described in Part II expects all corners to have thickness, and therefore to be detectable by the sensor. Under this supposition, it is easy to verify that any wedge-shaped object within range is visible if $\tau \geq 45$ deg (Figure 5(b)).

[1] If we add the restriction that the sensor range is also lower-bounded (objects cannot be detected below a range r_{min}), then the region s_l contains a hole and is not simply connected.

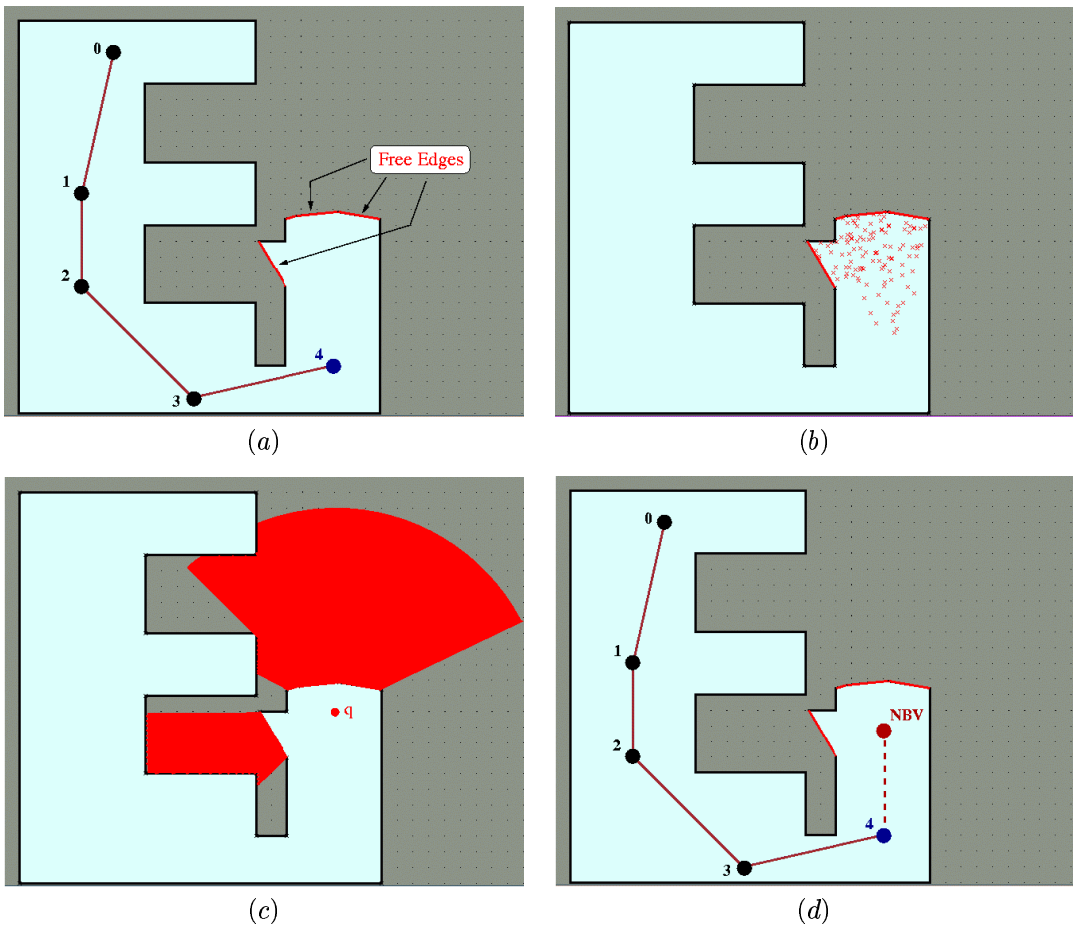


Figure 6: Steps in a next-best view computation: (a) safe region after 5 sensing operations; (b) only the regions within the visibility range of a free curve are sampled: (c) the potential visibility gain of a candidate q is the area $A(q)$ outside the explored region that can be visible *through* the free curves bounding $S_g(q_k)$; (d) the best candidate is selected as the maximizer of the quantity $A(q)\exp(-\lambda L(q, q_k))$.

3.3 Extracting Safe Regions from Real Sensor Data

The main practical difficulty with the results of this section is that the sensor output is usually a list of points, not a list of curves. A pre-processing stage must be added to the methods of this section to convert the raw data into the output list Π. The system described in Part II uses a geometric/numerical technique to convert a list of points into a list of polylines. Examples of this procedure are presented in Section 6.

4 A Next-Best View Algorithm

In a static environment, a safe region remains safe under the union operation. Hence, the layout model can be expanded iteratively. A first partial layout — a local safe region — is constructed from the data acquired by the range sensor at the robot's initial position q_0. At each iteration, the algorithm updates the layout model by computing the union of the safe region built so far with the local safe region generated at the new position q_k. The new safe region is then used to select the next sensing position q_{k+1}. To compute this next-best-view position, the procedure first generates a set of potential candidates. Next, it evaluates each candidate according to both the expected gain of information that will be sensed at this position, and the motion cost required to move there. These steps are illustrated in Figure 6, and described below.

4.1 Model Alignment and Merging

Let $M_g(q_{k-1}) = \langle \Pi_g(q_{k-1}), S_g(q_{k-1}) \rangle$ be the partial *global* model built at q_{k-1}. The term $S_g(q_{k-1})$ is the union of all local safe regions up to stage $k-1$. The boundary of $S_g(q_{k-1})$ is composed of free and solid curves, the latter representing physical sections of $\partial\mathcal{W}$. Let $\Pi_g(q_{k-1})$ be the list of solid curves in the boundary of $S_g(q_{k-1})$.

The robot performs a sensing operation once it moves into a new location q_k. From the local measurement, denoted as $\Pi_l(q_k)$, we can compute a local safe region $s_l(q_k)$ following the proof of Theorem 3.1. Let $m_l(q_k) = \langle \Pi_l(q_k), s_l(q_k) \rangle$ be the *local* model at q_k.

Suppose there exists an algorithm ALIGN that computes the transformation T aligning $m_l(q_k)$ with $M_g(q_{k-1})$ by matching the line segments of $\Pi_l(q_k)$ and $\Pi_g(q_{k-1})$. We will *not* assume that this technique is perfect: ALIGN computes a correct T only when there is enough overlap between $m_l(q_k)$ and $M_g(q_{k-1})$.

Once T is calculated, the new global safe region $S_g(q_k)$ is computed as the union of $T(S_g(q_{k-1}))$ and $s_l(q_k)$. The new model $M_g(q_k) = \langle \Pi_g(q_k), S_g(q_k) \rangle$ is represented in a coordinate frame centered over the robot at its current position q_k.

4.2 Candidate Generation

The future position q_{k+1} should potentially see large unexplored areas *through* the free curves bounding $S_g(q_k)$ (unexplored areas cannot be observed through solid curves because these represent sections of ∂W). However, we are constrained in our choices for q_{k+1}. The robot has to be entirely contained inside $S_g(q_k)$ at the next location q_{k+1}, which must also be reachable from q_k by a collision-free path. Furthermore, the function ALIGN must successfully find a transform T at the next position q_{k+1}. To achieve all these conditions, we proceed as follows:

Step 1. Randomly generate a set of possible next-best-view candidates $\mathcal{N}_{sam} \subset S_g(q_k)$ within the visibility range of the free curves bounding $S_g(q_k)$ (Figure 6(*b*)). This step prevents candidates from being unnecessarily created in areas that are far away from the free boundary of $S_g(q_k)$.

Step 2. For each $q \in \mathcal{N}_{sam}$, we compute the total length $\zeta(S_g(q_k), q)$ of the non-free curves bounding $S_g(q_k)$ that are visible from q under Definition 2.1 (see [19] for a survey of methods). ζ is a measure of the expected overlap between a new image $\Pi_l(q)$ and the updated history $\Pi_g(q_k)$. If $\zeta(S_g(q_k), q)$ is greater than some given threshold, then q remains a next-best-view candidate. This filtering stage ensures that the function ALIGN will successfully find a transform T.

Step 3. Suppose that there is a function $\mathcal{L}_{path}$ = PATH-PLANNER$(S_g(q_k), q_k, q)$ that computes a collision-free path (as a list of points) between q_k and q, or returns the empty list $\mathcal{L}_{path} = \emptyset$ if there is no such path. There are many implementation choices for PATH-PLANNER [12], and we make no assumptions about a particular one. To decide whether a position $q \in \mathcal{N}_{sam}$ remains a next-best-view candidate, we run the function PATH-PLANNER to verify if q is reachable from q_k. If q is not reachable, then such point is discarded.

After executing these three steps, we are left with a feasible set $\mathcal{N}_{sam}$ of NBV candidates.

4.3 Evaluation of Candidates

The score of every next-best-view candidate $q \in \mathcal{N}_{sam}$ is defined by the following function:

$$g(q) \quad = \quad A(q)\exp(-\lambda L(q, q_k)), \tag{1}$$

where λ is a positive constant, $L(q, q_k)$ is the length of the path $\mathcal{L}_{path}$ computed by the function PATH-PLANNER, and $A(q)$ is a measure of the unexplored area of the environment that is potentially visible from q (see below). q_{k+1} is selected as the sample $q \in \mathcal{N}_{sam}$ that maximizes $g(q)$.

The constant λ weights the relative cost of motion with respect to the potential visibility gain. $\lambda = 0$ implies that motion is "cheap", and the NBV algorithm is allowed to select the next-best view based solely in terms of the potential visibility gain. If $\lambda \to \infty$, then motion becomes so expensive that only locations near q_k are selected, as long as they produce a marginal gain in visibility.

Computation of A(q) We measure the potential visibility gain of each candidate q as a function of the area $A(q)$ outside the current safe region that may be visible *through* the free curves bounding $S_g(q_k)$ (Figure 6(c)). For polygonal models, $A(q)$ can be computed by the same ray-sweep algorithm used to compute classic visibility regions [19], with the following modifications:

1. The sweeping ray may cross an arbitrary number of free edges before hitting a solid one. Therefore, the computation-time of the ray-sweep algorithm becomes $\mathrm{O}(n \log(n) + n\, k_{\mathrm{f}})$, where k_{f} is the number of free edges bounding $S_g(q_k)$.

2. The resultant visible region is cropped to satisfy the range restrictions of the sensor. This operation can be done in $\mathrm{O}(n\, k_{\mathrm{f}})$.

4.4 Termination Condition

If the boundary of $S_g(q_k)$ contains no free curves, the 2-D layout is assumed to be complete; otherwise, $S_g(q_k)$ is passed to the next iteration of the mapping process (Figure 6(d)). A weaker test is used in practice: stop when the length of any remaining free curve is smaller than a specified threshold. This termination predicate is better suited to handle complex environments.

4.5 Iterative Next-Best View Algorithm

The general NBV algorithm is summarized below. Example runs (both in simulations and in physical experiments) are described in Part II.

Algorithm *Iterative Next-Best View*

Input: 1.- The current partial model $M_g(q_{k-1})$ and a new sensing position q_k
2.- The local sensor measurement $\Pi_l(q_k)$
3.- An image alignment function $T = \textsc{align}(m_l(q_k), M_g(q_{k-1}))$
4.- A path planning function $\mathcal{L}_{path} = \textsc{path-planner}(S_g(q_k), q_k, q)$
5.- The visibility constraints $\{r_{max}, \tau\}$
6.- The number of samples m, and a weighting constant $\lambda > 0$

Output: A next-best view position q_{k+1}

1. Compute the local safe region $s_l(qk)$. Set $\mathcal{N}_{sam} = \emptyset$.
2. Compute $T = \textsc{align}(m_l(q_k), M_g(q_{k-1}))$, and the union $S_g(q_k) = s_l(q_k) \bigcup T(S_g(q_{k-1}))$. Set $M_g(q_k) = \langle \Pi_g(q_k), S_g(q_k) \rangle$, where $\Pi_g(q_k)$ is the list of solid curves bounding $S_g(q_k)$.
3. Repeat until the size of $\mathcal{N}_{sam}$ is greater or equal than m:
   (a) Randomly generate a sample point $q \in S_g(q_k)$ in the visible range of the free curves bounding $S_g(q_k)$.
   (b) Compute the length $\zeta(S_g(q_k), q)$ of the non-free curves in $S_g(q_k)$ that are visible from q. If this number is less than the threshold required by ALIGN, discard q and repeat Step 3.
   (c) Compute the path $\mathcal{L}_{path} = \textsc{path-planner}(S_g(q_k), q_k, q)$. If the path does not exist, discard q and repeat Step 3.
   (d) Compute the visibility gain $A(q)$ and the length of the path $\mathcal{L}_{path}\ (q_k, q)$. Add q to the list of samples $\mathcal{N}_{sam}$ and repeat Step 3.
4. Select the sample in $\mathcal{N}_{sam}$ that maximizes the function $g(q) = A(q) \exp(-\lambda L(q, q_k))$ as the next-best view q_{k+1}.

The order of the steps in the above algorithm was selected for clarity purposes. In an actual implementation it will be more efficient to reorder some of these steps.

Part II

In the first part of this article we introduced the concept of safe region, and described how it can be used to produce collision-free motions and next-best view locations under image-alignment considerations. These techniques, however, will not produce a map-builder robot on their own. In order to build the layout of a building, several other operations must also take place besides the computation of the next-best view. These operations include polyline generation, and model alignment and merging, functions which are presented here.

In this second part we describe the implementation of our map-building robotic system, including the specific form of the NBV algorithm embedded in our prototype. The system architecture and some sample experiments using our system are also described.

5 Polygonal Maps

Let us assume that the robot is equipped with a polar range sensor measuring the distance between the sensor's center-point and the objects in the environment along several rays regularly spaced in a horizontal plane at height h above the floor. The sensor driver converts these measurements into a list of points representing the cross-section of the environment at a height h in a coordinate system attached to the sensor. Figure 7(b) shows such points for a 180-deg field of view, with 0.5-deg spacing between every two consecutive rays, captured using a range sensor from Sick Optic-Electronic. We model this sensor using Definition 2.1. It is not difficult to add an angular parameter α representing the field-of-view of the sensor, if necessary.

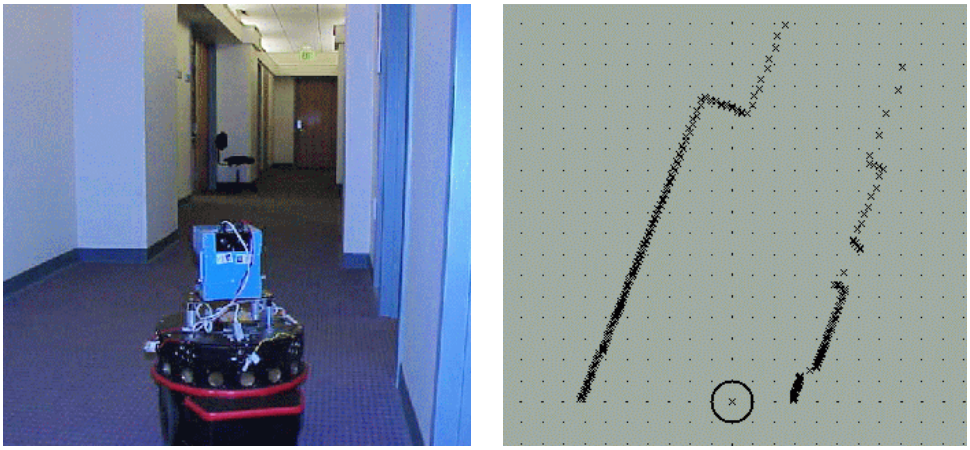
Our goal is to construct a polygonal layout of the environment from the sets of points captured by the range sensor at different locations. Our preference for polygonal models is driven by an algorithmic issue. Because of their compact representation, geometric properties can be computed very efficiently for polygonal models. For example, computing visibility regions for a polygonal model with n edges can be easily done in $O(n \log n)$ time with a classical line-sweep technique. Other representations, such as occupancy grids, do not allow such efficient computation.

Polygonal map-builders have traditionally been avoided in practical SLAM systems. In the past, sonars were the sensors of choice for gathering range data with a robot. But sonars are heavily affected by noise and provide low resolution data, so polygonal maps were out of the question. For this reason, most of the past research on map building has centered on constructing topological maps (which contain little geometric information) and occupancy grid representations (including the popular *probabilistic occupancy grid*, where probabilities are used to encode the uncertainty of an obstacle's presence). But range-finding technology has become cheaper, more precise and reliable. This has motivated us to reconsider polygonal representations.

6 Construction of 2-D Layouts

Ideally, the 2-D layout of an indoor environment is the projection — into a horizontal plane — of the portions of objects in the workspace that either block the robot's motion, and/or obstruct the field of view of its sensors. However, due to sensor limitations, the layout is often approximated as a horizontal cross-section cutting through the environment at a given height. In the case of a multi-floor environment, the model may consist of several 2-D layouts connected by passage-ways.

The robot builds a polygonal layout by moving through a sequence of sensing positions, which are chosen by the NBV planner described in Section 4. These positions could also be chosen by another software module or by a human operator. Independently on how the sensing locations are selected, the following steps are executed at each sensing position q: Polyline generation, safe region computation, model alignment, model merging, and detection of small obstacles. These are described in detail below.



(a) (b)

Figure 7: Range sensing: (a) scene; (b) captured points using a range sensor from Sick Optic-Electronic.

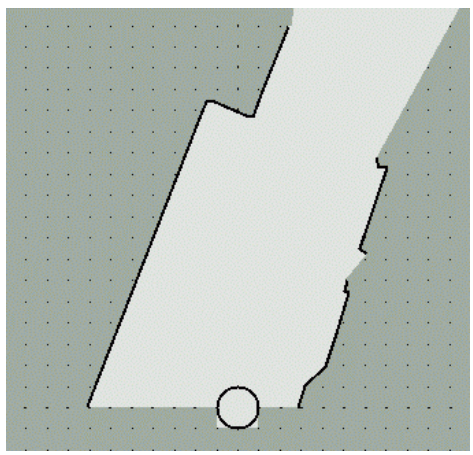
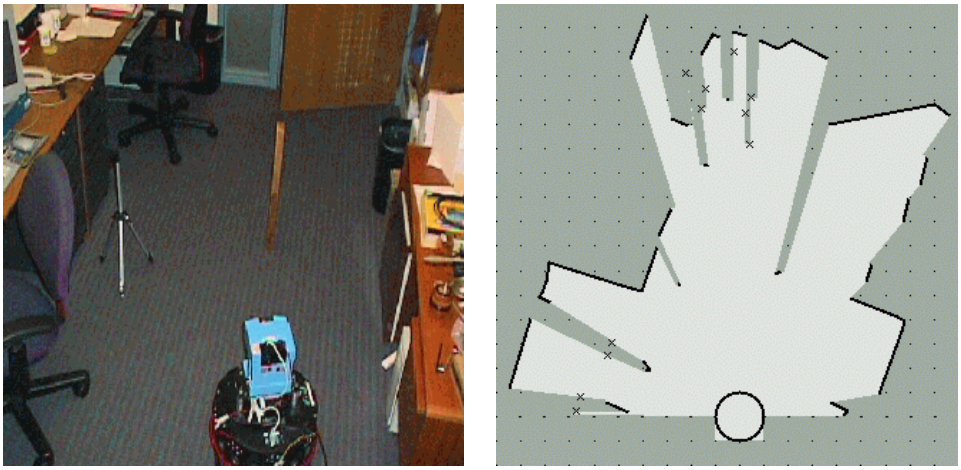


Figure 8: Polyline fit for the data of Figure 7(b).

6.1 Polyline Generation

Let L be the list of points acquired by the sensor at q. L is transformed into a collection Π_l of polygonal lines called *polylines*. The polyline extraction algorithm operates in two steps: (1) group data into clusters, and (2) fit a polyline to each cluster. The goal of clustering is to group points that can be traced back to the same surface object. A sensor with infinite resolution would capture a curve $r(\theta)$ instead of a sequence of points. This curve would be discontinuous at exactly the points where occlusions occur. For a real sensor, discontinuities are detected using thresholds selected according to the sensor's accuracy.

The points in each cluster are fitted with a polyline so that every data point lies within a distance ϵ from a line segment, while minimizing the number of vertices in the polyline. The computation takes advantage of the fact that the data delivered by our polar sensor satisfy an ordering constraint along the noise-free θ-coordinate. By applying the mapping $u = \cos\theta/\sin\theta, v = 1/(r\sin\theta)$, the problem is transformed into a linear fit of the form $v = a + bu$ (which maps to $bx + ay = 1$ in Cartesian (x, y)-space). Several well-known recursive algorithms exist to find polylines in (u, v)-space [22]. By converting ϵ to the position-dependent error bound $e = \epsilon v\sqrt{(a^2 + b^2)}$ in the (u, v)-space, each data point in the (x, y)-space is guaranteed to be within ϵ from the computed polyline. Is important to note that the polyline fitting process also acts as a noise reduction filter.



(a) (b)

Figure 9: A more complicated example of a polyline fit.

Figure 8 shows a set of three polylines generated from the data points of Figure 7(*b*). A more complicated example, in a cluttered office environment, is displayed in Figure 9. The area in light grey in (*b*) is the robot's visibility region under the classical line-of-sight model (this region is not a safe region).

6.2 Safe Region Computation

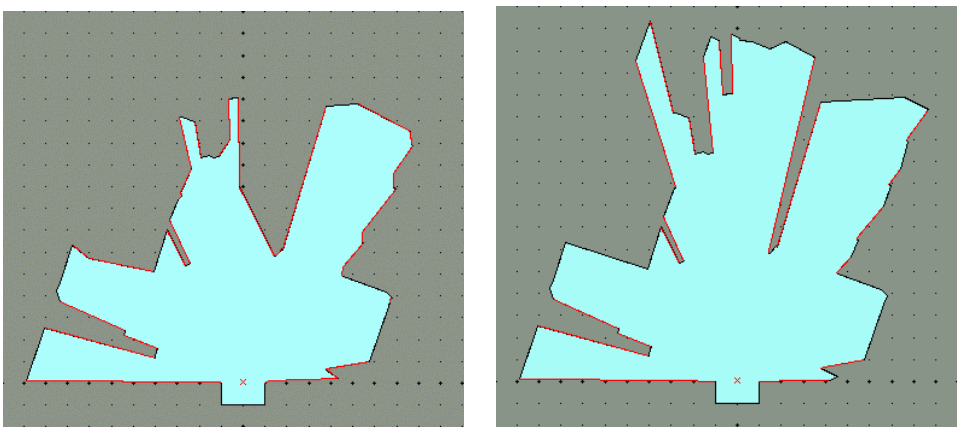
Once the polyline set Π_l is extracted from the data, a safe region $s_l(q)$ is computed. In Section 2, $s_l(q)$ was defined as the largest region guaranteed to be free of obstacles given the local observation $\Pi_l(q)$. This region is a function of the sensor's visibility model.

The local safe region $s_l(q)$ is bounded by both the observed polylines $\Pi_l(q)$, and by free curves connecting the polyline endpoints. Free curves can be computed *exactly* for polygonal workspaces, and are composed of segments, circular arcs and spiral sections. The construction of free curves is fully described in the proof of Theorem 3.1 included in the appendix. In our implementation, we approximated arcs and spiral sections with polygonal lines to simplify subsequent computations. Therefore, the region $s_l(q)$ is bounded by *solid edges* (derived from the observed polylines), and *free edges* (derived from the approximated free curves). Once the region $s_l(q)$ is computed, the pair $m_l(q) = \langle \Pi_l(q), s_l(q) \rangle$ becomes the *local* model constructed at location q.

Figure 10 shows two local safe regions computed for the scene in Figure 9 for different values of the maximal range r_{max} and the incidence angle τ.

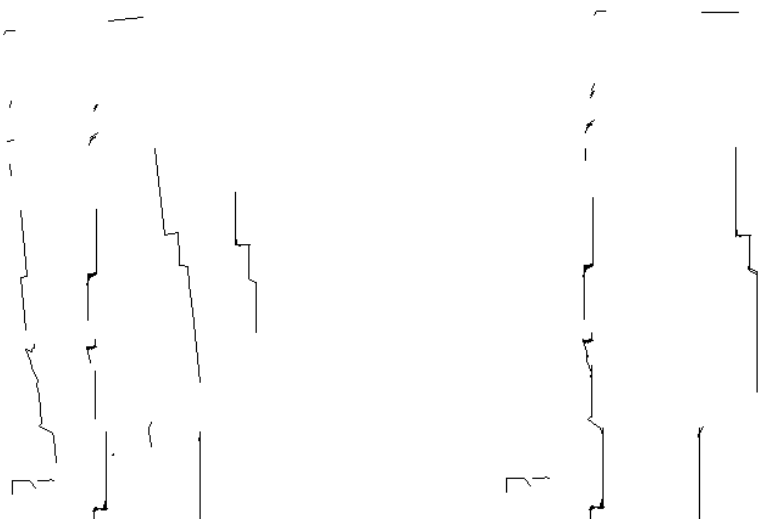
6.3 Model Alignment

Let $M_g(q_{k-1}) = \langle \Pi_g(q_{k-1}), S_g(q_{k-1}) \rangle$ be the partial *global* model built at location q_{k-1}. Assume for the time being that such global model exists. Suppose now that the robot executes a new sensing operation at location q_k, extracting the local model $m_l(q_k) = \langle \Pi_l(q_k), s_l(q_k) \rangle$. A best match is then computed between the line segments in $\Pi_g(q_{k-1})$ and those in $\Pi_l(q_k)$, yielding an Euclidean transform aligning both sets of polylines. The matching algorithm is similar to a previous technique used to discover and align substructures shared by 3-D molecular structures [8]. The algorithm selects pairs of line segments from Π_l at random. For each pair (u_1, u_2), it finds a pair of segments (v_1, v_2) in Π_g with the same relative angle. The correspondence $u_1 \rightarrow v_1,\ u_2 \rightarrow v_2$ yields a transform $T(x, y, \theta)$ obtained by solving least-square equations. The algorithm then identifies the segments of Π_l and Π_g which match under this transform, and creates a new correspondence $u_1 \rightarrow v_1,\ u_2 \rightarrow v_2,\ \ldots,\ u_r \rightarrow v_r$, where the u_i's and v_i's are not necessarily distinct. It recalculates the transform based on this new correspondence and evaluates the quality of fit. The above steps are repeated for each



(a) (b)

Figure 10: Computed safe regions: (a) $r_{max} = 275$ cm and $\tau = 50$ deg; (b) $r_{max} = 550$ cm and $\tau = 85$ deg.



(a) (b)

Figure 11: (a) Unaligned polylines; (b) computed alignment.

pair of line segments sampled from Π_l, and the transform with the best quality is retained. If all segments in Π_l are approximately parallel, the algorithm uses endpoints and odometric sensing to approximate the missing parameter of the transform.

Figure 11(a) shows two sets of polylines before alignment. The computed alignment of these two sets is displayed in (b).

6.4 Model Merging

The selected transform T is applied to $S_g(q_{k-1})$, and the new global safe region $S_g(q_k)$ is computed as the union of $T(S_g(q_{k-1}))$ and $s_l(q_k)$. The solid edges bounding $S_g(q_k)$ form the new polyline set $\Pi_g(q_k)$. To avoid edge fragmentation, consecutive solid (respectively free) edges in the boundary of $\Pi_g(q_k)$ that are practically collinear under the sensor's resolution are fused into a single edge. The new model $M_g(q_k) = \langle \Pi_g(q_k), S_g(q_k) \rangle$ is represented in the coordinate system attached to the robot at its current position q_k. The 2-D layout is considered complete if no remaining free edge in the boundary of $S_g(q_k)$ is longer than a given threshold; otherwise, $S_g(q_k)$ is passed to the next iteration of the mapping process.

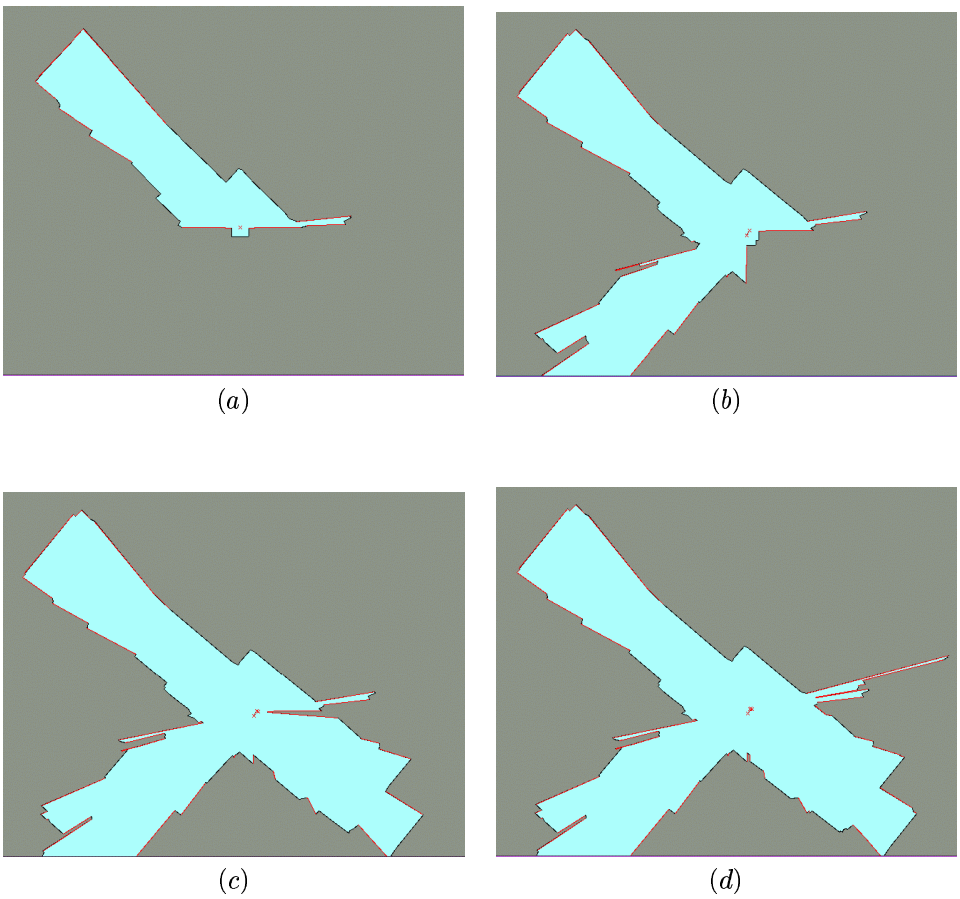


Figure 12: Merging four partial models at a given position.

Figure 12 displays four partial models. The robot is at some location where it rotates to face four successive directions spaced by 90 deg. The local model in (a) was built at the first orientation. The model in (b) was obtained by merging the model of (a) with the local model generated at the second orientation, and so on. The model in (d) combines the data collected at the four orientations. In addition to being an illustration of model merging, Figure 12 shows the artifice employed to emulate omnidirectional sensing using a sensor with a 180-deg field of view. Note that the matching operation compensates for any small variation in the robot's position as it rotates to a new orientation.

6.5 Dealing with Small Obstacles and Transient Objects

A horizontal cross-section through an indoor environment often intersects small objects (e.g., chair legs). Such objects are detected by a good range sensor and hence appear among the polylines $\Pi_l(q)$ extracted at a sensing position q (see Figure 9). But, due to small errors in aligning polylines, such obstacles tend to be eliminated when the union of two safe regions is computed. Other model merging techniques could be used, but modeling small obstacles by closed polygonal contours is known to be difficult and not very reliable. In many instances, a map is more useful when small obstacles are omitted, since the positions of such obstacles often tend to change over time. So, we proceed as follows. Small obstacles result into narrow "spikes" pointing into the safe region $s_l(q)$ obtained at q. These spikes can be automatically detected. The apex of each detected spike is a small isolated polyline, which is saved in a separate *small-object map*. Hence, the final model consists of a main layout (polylines and safe region) and a secondary map (small-object map). Figure 13 shows the small obstacles (apexes enclosed by square boxes) detected in the scan from Figure 9. These include an aluminum camera tripod, a narrow wooden bar, and a swivel chair.

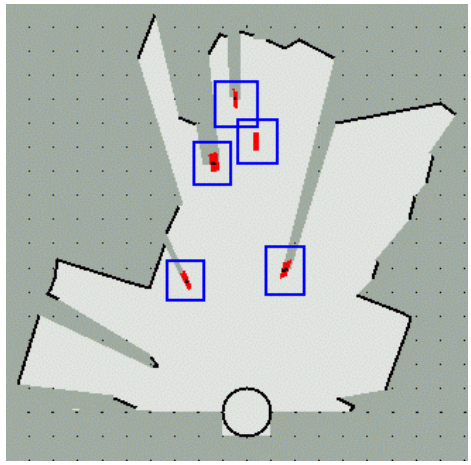


Figure 13: Small obstacles extracted from Figure 3(b).

Merging partial models by taking the union of safe regions has the added advantage of eliminating transient objects. Comparing edges in successive partial models allows detection of such objects, which can be recorded in a separate structure (in a way similar to small objects). However, this capability is not implemented in the system described here.

7 NBV Planner Implementation

The implementation of the on-line NBV planner is based upon the algorithm described in Section 4. The basic algorithm is applied here essentially unchanged: (1) the robot moves to a position q_k and takes a new observation; (2) a local safe region $\Pi_l(q_k)$ is computed; (3) the partial layout model $M_g(q_k)$ is updated; (4) a set of next-best-view candidates is generated; and (5) the best candidate is selected as the next-best view.

The operations involved in steps 1-3 are polyline extraction, safe region computation, and model alignment and merging. These operations were described in the previous section. The general principles behind steps 4 and 5 were explained in Section 4.2 and 4.3, respectively. The implementation details of these steps are described in this section.

7.1 Candidate Generation

A position q in $S_g(q_k)$ is a good next-best-view candidate if it can potentially see large unexplored areas of the workspace. Section 4.2 proposed to consider only those points within the visibility range of the free edges bounding $S_g(q_k)$ as potential candidates — see Figure 6(b). In our implementation, the set of possible next-best-view candidates $\mathcal{N}_{sam}$ is randomly generated using the following procedure:

RANDOM GENERATION OF SAMPLES

1. Select positive constants σ and ρ, representing samples per unit length and samples per unit area, respectively.
2. For each free edge e in the boundary of $S_g(q_k)$, uniformly select $\sigma\times$length$[e]$ random points along e. Group these boundary samples under the set $\mathcal{B}$.
3. For each point $p \in \mathcal{B}$, compute the visibility region $\mathcal{V}(p)$ inside $S_g(q_k)$, upper-limited to a range r_{max} from p, and uniformly select $\rho\times$area$[\mathcal{V}(p)]$ random points inside $\mathcal{V}(p)$. Group all the interior sample points under the set $\mathcal{N}_{sam}$.

The computational cost of this procedure is $\mathrm{O}((kn+m)\log n)$, where k is the total number of boundary samples, m the number of generated candidates, and n the number of edges in the partial layout $S_g(q_k)$.

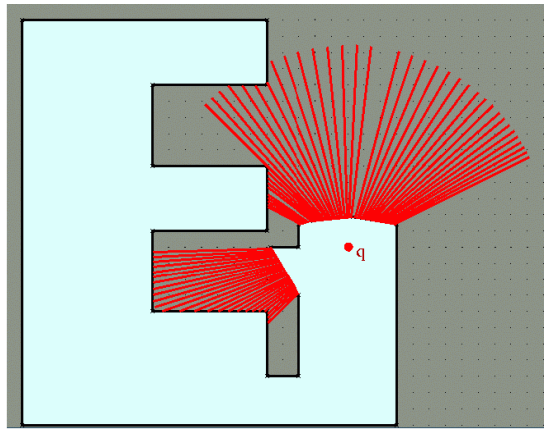


Figure 14: The potential visibility gain of a candidate q is the area $A(q)$ outside the explored region that can be visible through the free edges bounding $S_g(q_k)$, discretized here by casting rays.

Next, we must verify that the samples in $\mathcal{N}_{sam}$ are indeed feasible. As explained in Section 4.2, a candidate position q should not only be contained inside $S_g(q_k)$, but it must also be reachable from q_k by a collision-free path. For circular robots, the implementation of the function $\mathcal{L}_{path}$ = PATH PLANNER$(S_g(q_k), q_k, q)$ can be rather straightforward:

PATH PLANNING

1. Shrink $S_g(q_k)$ by the radius of the robot [12] (this operation can be done in linear time). Shrinking is performed only when the robot moves to a new sensing position q_k, not every time a candidate q is evaluated.

2. Compute the shortest path from q_k to a q inside the shrunk region. This computation can done in $\mathrm{O}(n^2)$ per query using the visibility graph method [12]. The query time, however, can be improved to $\mathrm{O}(n)$ by constructing the *shortest-path map* from q_k — $\mathrm{SPM}(q_k)$. This map is a decomposition of the space into cells, such that the shortest paths to q_k from all points within a cell share the same sequence of obstacle vertices. The map $\mathrm{SPM}(q_k)$ can be constructed in $\mathrm{O}(n^2)$ of preprocessing time, although faster and more complicated schemes exist (see [17] for a survey of methods).

Finally, for each candidate q we compute $\zeta(S_g(q_k), q)$: the total length of the solid edges in the boundary of $S_g(q_k)$ that are visible from q under the sensor's range and incidence constraints. ζ is the measure of the expected overlap between a future image $\Pi_l(q)$ and the current history $\Pi_g(q_k)$. If the computed overlap is greater than some threshold, then q remains a feasible candidate. Otherwise, q is discarded. This filtering stage ensures that the line matching algorithm will work reliably at the next position q_{k+1}, and the computational cost is only $\mathrm{O}(n \log n)$ per query (using a ray-sweep algorithm).

7.2 Evaluation of Candidates

Every next-best-view candidate q is evaluated using the ranking function proposed in Section 4.3:

$$g(q) \quad = \quad A(q) \exp(-\lambda L(q, q_k)). \tag{2}$$

Due to our choice of planner, here $L(q, q_k)$ is simply the length of the shortest path connecting q_{k-1} with q. The parameter λ was set to 20 cm^{-1} in the implementation, a value that prevents the robot from oscillating back and forth between regions with similar visibility potential.

As for the estimation of the visibility gain $A(q)$, Section 4.3 proposes that it be the area of the unexplored region visible from q through the free edges in the boundary of $S_g(q_k)$ (imagine the free edges are windows). In our implementation, $A(q)$ is computed using a discretized version of the technique outlined in Section 4.3:

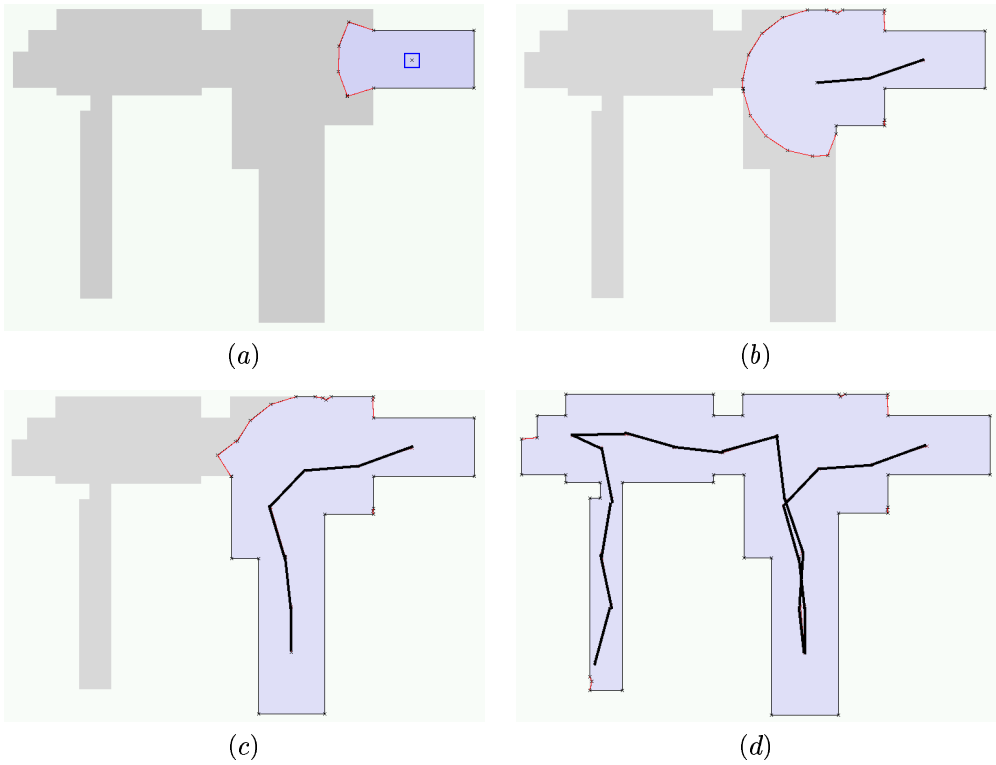


Figure 15: Example 1 of model construction in simulation.

COMPUTATION OF $A(q)$

1. A fixed number of equally spaced rays is casted from q.
2. Along each ray, consider only the segment within 0 and r_{max} from q. If the segment intersects one or more solid edges, we eliminate the portion beyond the first of these intersections.
3. For each remaining ray section, compute the length ℓ of the portion falling *outside* the region $S_g(q_k)$.
4. Finally, estimate $A(q)$ as the sum of all the computed ℓ's.

The estimation of $A(q)$ is illustrated in Figure 7.1. The sum of all shown radial sections (the ℓ's) is an approximation of the area beyond the free boundary of $S_g(q_k)$ that is visible from q.

7.3 Example Runs

Figure 15 shows partial models generated at several iterations (0, 2, 6, and 19) during a run of the planner on simulated data, and the path followed by the robot. The layout model was completed in 19 iterations. Because path length is taken into account in the goodness function, the robot fully explores the bottom-right corridor before moving to the left corridor. Figure 16 shows another series of snapshots for the same environment, the same initial position, but greater r_{max} and τ. The motion strategy is simpler, requiring only 7 iterations.

8 System Architecture and Experiments

The map-building robot system was implemented on a Nomadic SuperScout wheeled platform. The robot was equipped with a laser range sensor from Sick Optic-Electronic (Figures 7(a) and 9(a)). The sensor uses a

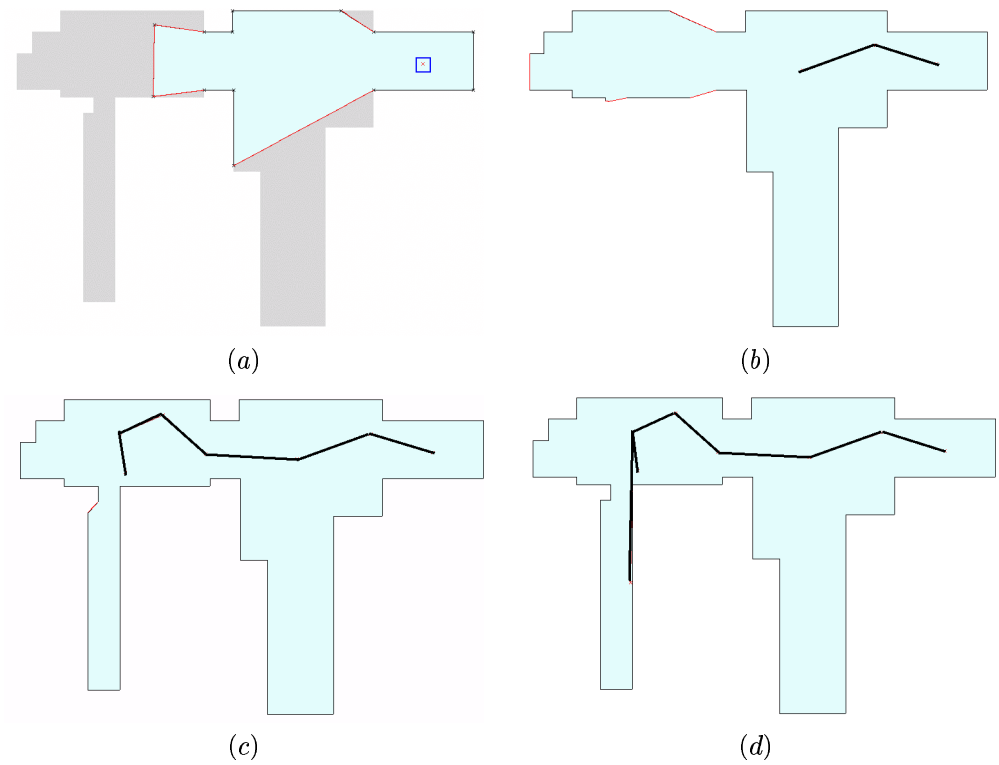


Figure 16: Example 2 of model construction in simulation.

time-of-flight technique to measure distances. The robot's on-board processor (Pentium 233 MMX) acquires a 360-point 180-deg scan in 32 ms through a SeaLevel 500 Kbs PCI serial card. At each sensing location, a 360-deg view is obtained by taking 4 scans (Figure 12).

The on-board processor is connected to the local-area network via 2 Mbs radio-Ethernet. The NBV planner and the navigation monitor run off-board in a Pentium II 450 MHz Dell computer. The software was written in C++ and uses geometric functions from the LEDA-3.8 library [16].

8.1 System Architecture

The software consists of several modules executing specialized functions, communicating with each other through TCP/IP socket connections under a client/server protocol. These modules are shown in Figure 17.

A *Sick sensor server* handles communications with the SeaLevel card. It allows clients to assume that they are connecting to a device resembling an ideal sensor. The server offers the following capabilities:

- choice among 3 speed modes: 1, 5, and 30 scans/sec,
- batch transmission of multiple scans on request,
- scan averaging using the sensor's on-board electronics,
- operation in continuous mode,
- real-time polyline fitting with 2.5-cm error bound.

Since the polyline fitting technique is fast enough to be performed in real time under any speed mode, it is embedded into the server's code. This reduces the amount of data transmitted to the clients.

A *navigation monitor* allows a user to supervise the exploration process. The user may query the NBV module for the next position and/or the most recent environment model, or select the next sensing position manually. The user may also teleoperate the robot in continuous mode, receiving scan updates every 0.1 sec. The navigation module is also responsible for aligning new data with the previous model. The module

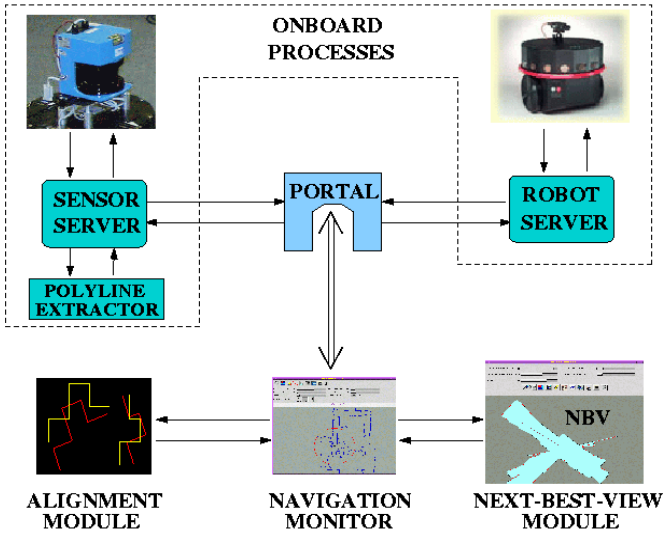


Figure 17: Interaction among the modules of the next-best-view system.

first pre-aligns new data using the robot's odometry, and it afterwards invokes the model matching function. The computed transform is sent to the NBV module with each new scan.

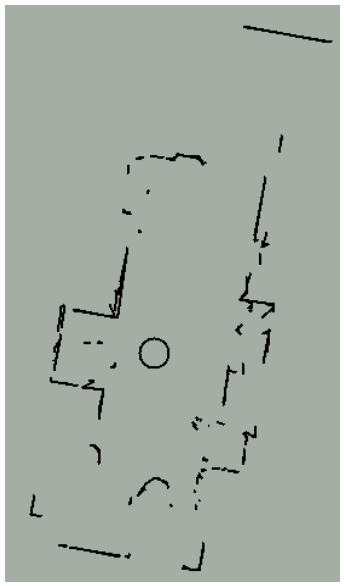
Finally, an *NBV module* computes the next position given the current model of the world. The model is updated every time a new scan is received.

8.2 Layout Construction with the Robotic System

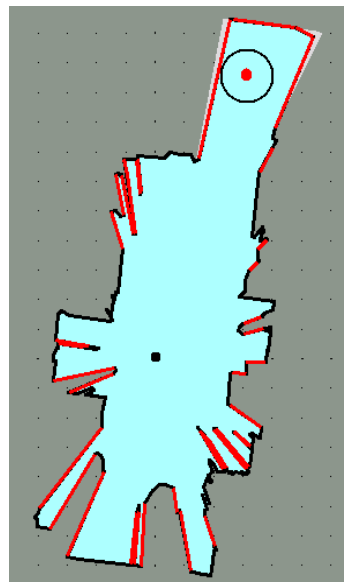
Figure 18 shows a sequence of partial layouts built by the map-building robot system in our laboratory. The robot is initially placed in a messy office with many small obstacles (chair and table legs, and cables). The sensor parameters are $r_{max} = 550$ cm and $\tau = 85$ deg. The polylines extracted at this initial location are shown in (a), and the safe region is displayed in (b) along with the next sensing position computed by the NBV planner. The safe region is bounded by many free edges forming spikes, but the candidate evaluation function automatically detects that little additional space can be seen through such free edges. Consequently, the NBV planner reliably selects the next sensing position near the exit door of the office. Figures (c)-(e) show the safe region after the robot has reached the second, fourth and sixth sensing positions, respectively. At that stage, the layout consists of the initial office, a second office (incompletely mapped), and two perpendicular corridors.

A longer run is shown in Figure 19, where the robot mapped a section of the Robotics Lab. at Stanford University. The first 6 iterations are shown in (a). At this point the executed strategy resembles the one shown in Figure 18 due to similar initial conditions. At the corridor intersection, however, the robot faces more choices than in the previous example because an office door is open. Nevertheless, the planner opted to continue moving along a corridor, all the way into the other hall (b). Glass is transparent to the sensor's laser, so the robot failed to detect the glass door indicated in (b). At this point, the operator overrode the decision of the NBV planner, who interpreted the vicinity of the glass door as the threshold of an unexplored open area. Finally, in (c), the robot moved down the second hall until it reached the lab's lounge. The planner decided then to send the robot to explore this newly detected open area.

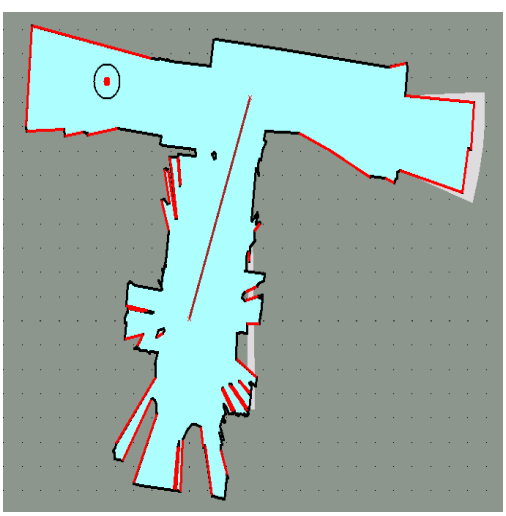
Figure 20 shows all the observed polylines after the robot completed a circuit around the lab. The polylines shown in light color (red) were captured at the last location, and the area inside the circle is the range of the last scan. Note the final mismatch. This discrepancy appears because matching transforms are computed *locally* to align the current view $\Pi_l(q_k)$ with the current history $\Pi_g(q_{k-1})$. Once a transform is computed, it is never revised. The mismatch occurs due to the lack of a global optimization function. Although this is not a failure of our NBV method, it is a limitation of our current implementation.



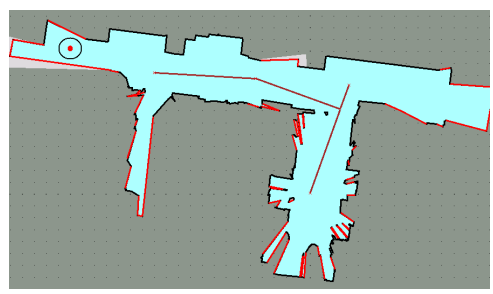
(a)



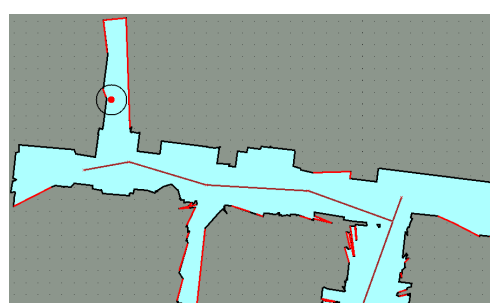
(b)



(c)



(d)



(e)

Figure 18: Experiments using the integrated robotic system.

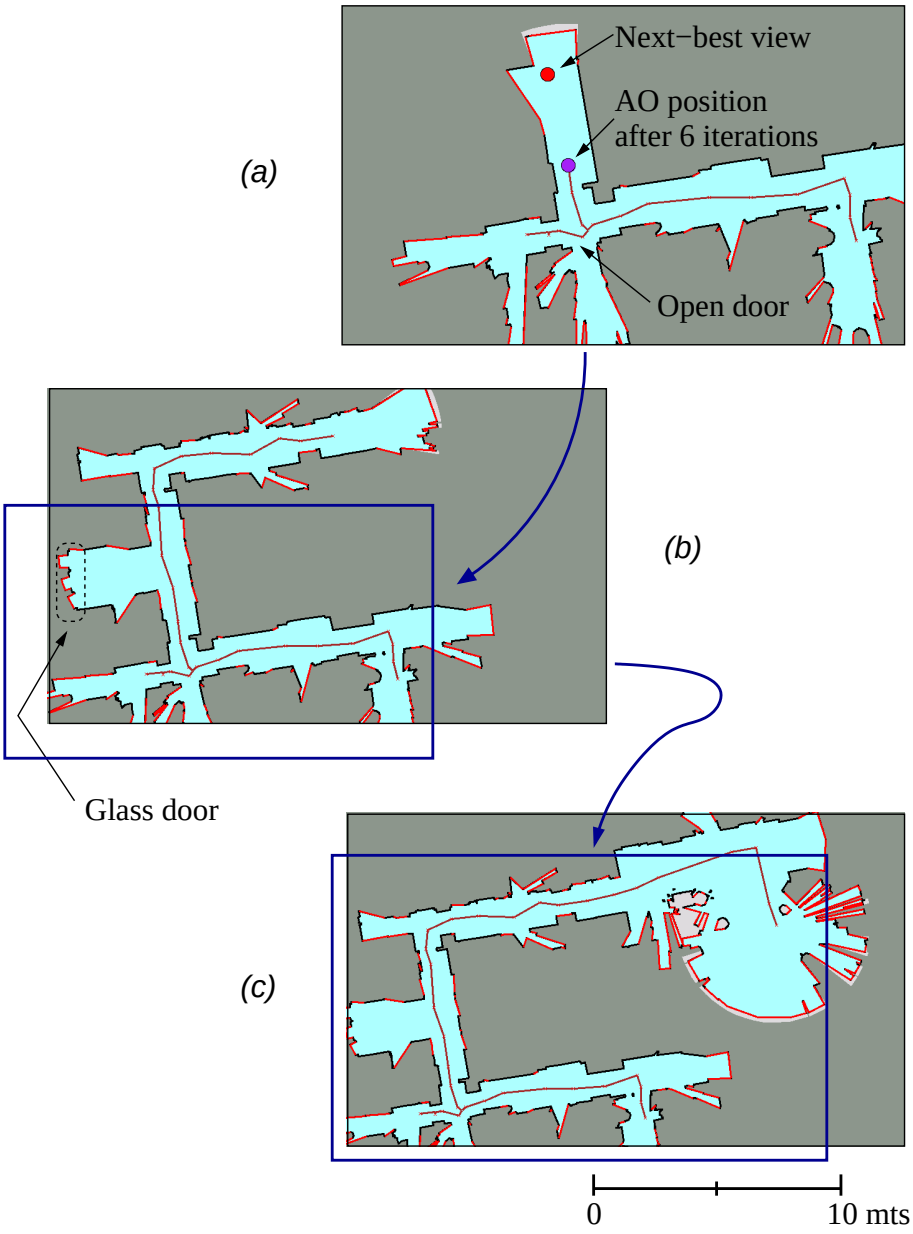


Figure 19: A run around the Robotics Lab. at Stanford University (Gates Building, wing 1-A): (a) the map built after 6 iterations — due to similar initial conditions, the strategy at this point resembles that one from Figure 18; (b) the robot moved up into a second hall, after the user overrode a plan to move toward a glass door; (c) the robot maps the second hall until it reached an open area, which the robot decided to explore next.

Figure 20: Observed polylines after the robot has completed a tour around the Robotics Lab. The polylines shown in light color were captured in the last location. The region inside the circle shows the final mismatch that results from relying exclusively on local matches.

9 Results and Limitations

The experiments with the robot system, both in simulated and real environments, show that the NBV planner can considerably reduce the number of motions and sensing operations required for map building. This claim is difficult to quantify, as it requires of more extensive comparisons between our plans and strategies produced by other means (e.g., trained human operators). Our experiments, however, have so far shown that the planner produces strategies that cannot be easily be out-done by a human operator. Our system also demonstrates that polygonal maps are feasible representations for indoor map-building tasks. As it was argued in Section 5, polygonal models offer significant advantages over other representations.

The most obvious limitation of our map-building system is that it only builds a cross-section of the environment at a fixed height. Therefore, important obstacles may not be sensed. One way to improve sensing is to emit laser rays at multiple heights. The techniques described in this article would still remain applicable without significant changes.

A more fundamental limitation is the lack of error-recovery capabilities. Any serious error in polyline extraction or during image alignment can result in a completely unacceptable model. For that reason, the overall prototype design is very conservative. For instance, often many points delivered by the sensor are discarded to avoid generating incorrect polylines. This may lead the robot to sense the same part of an environment several times, hence producing a longer motion path. Moreover, the image-alignment function always runs under user supervision to avert serious registration errors.

Matching transforms are computed locally to align the global model with the local model generated at the robot's current location. Once a transform has been computed, it is never revised. This has its disadvantages. In corridors bounded by parallel featureless walls, line matching only corrects positioning errors in the direction perpendicular to the walls. Odometry can be used, but imprecision in the direction parallel to the walls grows bigger with distance. A possible solution to this problem is to track successive local models and transformations to enable the periodic optimization of a global matching criterion, especially after the robot has completed a long loop around a building. This could be achieved by adapting some of the localization techniques recently proposed in the SLAM literature [6, 14]. For large environments such global optimization will produce more precise layouts, and solve mismatches like the one shown in Figure 20.

Finally, the current system should handle multiple robots with relatively minor changes. If a team composed of N robots is available, and their relative positions are known, a single model can be generated from all the captured scans. A central NBV planner then computes the set of N positions that stations the

team for the aggregated next-best view. However, when the relative positions of the robots are not known, the problem becomes considerably more difficult. In this case, the robots act independently (distributed planning), and perhaps communicate only sporadically. The techniques presented in this article have to be revised and extended in order to cover this case.

Acknowledgments: This work was funded by DARPA/Army contract DAAE07-98-L027, ARO MURI grant DAAH04-96-1-007, NSF grant IIS-9619625, and a gift from Honda R&D, Americas. We also wish to thank L. Guibas, T.M. Murali and R. Murrieta, whose contributions enriched the ideas presented in this article, and S. Yao and E. Mao for their assistance during the implementation of the robot system.

References

[1] J.E. Banta, Y. Zhien, X.Z. Wang, G. Zhang, M.T. Smith, and M.A. Abidi. A "next-best-view" algorithm for three-dimensional scene reconstruction using range images. In *SPIE*, volume 2588, pages 418–29, 1995.

[2] R. Chatila and J.P. Laumond. Position referencing and consistent world modeling for mobile robots. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 138–143, 1985.

[3] H. Choset and J. Burdick. Sensor based motion planning: The hierarchical generalized voronoi diagram. In J.-P. Laumond and M. Overmars, editors, *Proc. 2nd Workshop on Algorithmic Foundations of Robotics*. A.K. Peters, Wellesley, MA, 1996.

[4] C. I. Conolly. The determination of next best views. In *IEEE Int. Conf. on Robotics and Automation*, pages 432–435, 1985.

[5] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. ACM SIGGRAPH*, pages 303–312, August 1996.

[6] H.F. Durrant-Whyte, M.W.M.G. Dissanayake, and P.W. Gibbens. Toward deployment of large scale simultaneous localisation and map building (slam) systems. In J. Hollerbach and D. Koditschek, editors, *Robotics Research - The Ninth Int. Symp.*, pages 161–167, New York, NY, 2000. Springer.

[7] A. Elfes. Sonar-based real world mapping and navigation. *IEEE J. Robotics and Automation*, RA-3(3):249–265, 1987.

[8] P.W. Finn, L.E. Kavraki, J.C. Latombe, R. Motwani, C. Shelton, S. Venkatasubramanian, and A. Yao. Rapid: Randomized pharmacophore identification for drug design. *J. of Comp. Geometry: Theory and Applications*, 10:263–272, 1998.

[9] H. González-Banos, A. Efrat, J.C. Latombe, E. Mao, and T.M. Murali. Planning robot motion strategies for efficient model construction. In J. Hollerbach and D. Koditschek, editors, *Robotics Research - The Ninth Int. Symp.*, pages 345–352, New York, NY, 2000. Springer.

[10] H. González-Banos and J.C. Latombe. Planning robot motions for range-image acquisition and automatic 3d model construction. In *Proc. AAAI Fall Symposium Series, Integrated Planning for Autonomous Agent Architectures*, Orlando, Florida, October 23-25 1998. AAAI Press.

[11] B. Kuipers, R. Froom, W.K. Lee, and D. Pierce. The semantic hierarchy in robot learning. In J. Connell and S. Mahadevan, editors, *Robot Learning*. Kluwer Academic Publishers, Boston, MA, 1993.

[12] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[13] J.J. Leonard and H.F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991.

[14] J.J. Leonard and H.J.S. Feder. A computationally efficient method for large-scale concurrent maping and localization. In J. Hollerbach and D. Koditschek, editors, *Robotics Research - The Ninth Int. Symp.*, pages 169–176, New York, NY, 2000. Springer.

[15] J. Maver and R. Bajcsy. Occlusions as a guide for planning the next view. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(5):417–433, May 1993.

[16] K. Mehlhorn and St. Nahër. *LEDA: A Platform of Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 1999.

[17] J.S.B. Mitchell. Shortest paths and networks. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 445–466. CRC Press, Boca Raton, FL, 1997.

[18] P. Moutarlier and R. Chatila. Stochastic multisensory data fusion for mobile robot location and environment modeling. In H. Miura and S. Arimoto, editors, *Robotics Research - The 5th Int. Symp.*, pages 85–94. MIT Press, Cambridge, MA, 1989.

[19] J. O'Rourke. Visibility. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 467–479. CRC Press, Boca Raton, FL, 1997.

[20] R. Pito. A solution to the next best view problem for automated cad model acquisition of free-form objects using range cameras. Technical Report 95-23, GRASP Lab, University of Pennsylvania, May 1995.

[21] R. Pito. A sensor based solution to the next best view problem. In *Proc. IEEE 13th Int. Conf. on Pattern Recognition*, volume 1, pages 941–5, 1996.

[22] W.H. Press, S.A. Teukolsky, W.T. Vettering, and B.P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1994.

[23] S. Teller. Automated urban model acquisition: Project rationale and status. In *Proc. 1998 DARPA Image Understanding Workshop*, pages 455–462. DARPA, 1998.

[24] S. Thrun, D. Fox, and W. Burgard. Probabilistic mapping of an environment by a mobile robot. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1546–1551, Leuven, Belgium, 1998.

[25] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proc. ACM SIGGRAPH*, pages 311–318, 1994.

[26] L. Wixson. Viewpoint selection for visual search. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 800–805, 1994.

A The Complexity of a Free Curve is $\mathbf{O}(1)$

This appendix demonstrates that the complexity of a free curve is constant. Specifically, we provide here the proof for Theorem 3.1:

Theorem 3.1 (Free Curves) *Suppose $r_2(\theta; a_2, b_2)$ succeeds $r_1(\theta; a_1, b_1)$ in the output list Π of a sensor operating under of a sensor operating under Definition 2.2 and located at the origin. If $\partial\mathcal{W}$ is continuously differentiable, then the free curve $f(\theta; b_1, a_2)$ connecting r_1 to r_2 consists of at most three pieces. Each piece is either a line segment, a circular arc, or a section of a logarithmic spiral of the form $r = r_o \exp(\pm\lambda\theta)$ (where $\lambda = \tan\tau$ and r_o is a constant).*

In order to prove this claim we need the following lemma:

Lemma A.1 (Unobserved Obstacles) *Let $r_2(\theta; a_1, b_1)$ succeed $r_2(\theta; a_2, b_2)$ in the list Π. Let C be some obstacle, and suppose that neither r_1 nor r_2 are part of the boundary of C (i.e., C is disjoint from r_1 and r_2). If $\partial\mathcal{W}$ is continuously differentiable, then no portion of C lies within a distance r_{max} from the origin in the polar interval $b_1 < \theta < a_2$.*

Proof: Suppose the lemma is not true — that is, there is a portion of C within r_{max} of the origin inside the polar interval (b_1, a_2). Let p be the closest point to the origin in the boundary of C. Because $\partial\mathcal{W}$ is differentiable, the normal of $\partial\mathcal{W}$ at p points toward the origin. Therefore, p and its vicinity should have been observed. The vicinity of p must then be part of an element of Π. But this contradicts our assumption that r_2 succeeds r_1 and that C is disjoint from r_1 and r_2. $\square$

The consequence of Lemma A.1 is that if there exists an obstacle (or a portion of an obstacle) within the sensor's range inside the polar interval (b_1, a_2), then r_1 and/or r_2 represent a portion of this obstacle's boundary. In other words, in order to construct the worst-case scenario in the polar sector (b_1, a_2), we can assume that the workspace has no holes, and consider r_1 and r_2 as boundary sections of the same obstacle.

From here on, let $\beta = a_2 - b_1$, $\rho_1 = r_1(b_1)$ and $\rho_2 = r_2(a_2)$; and let l_1 and l_2 denote the rays connecting the origin with point $p_1 = (\rho_1, b_1)$ and point $p_2 = (\rho_2, a_2)$, respectively.

Each endpoint of a curve in Π represents one of the following events: the sensor line-of-sight was occluded (denoted as case {o}), the range constraint was exceeded (case {e}), or the incidence constraint was exceeded (case {v}). To join p_1 with p_2 there are a total of 6 distinct cases: {v,v}, {v,o}, {v,e}, {e,e}, {o,o} and {e,o}. The cases {o,e}, {o,v} and {e,v} are mirror images of other cases.

Case {v,v}: The incidence constraint was exceeded immediately after $\theta = b_1$ and immediately before $\theta = a_2$. Therefore, the normal to $\partial\mathcal{W}$ just after r_1 and just before r_2 is oriented at a grazing angle with respect to the sensor. Suppose that the boundary $\partial\mathcal{W}$ continues after r_1 with its surface normal constantly oriented at exactly an angle τ with respect to the sensor's line-of-sight. This curve in polar coordinates satisfies the following relations:

$$\boldsymbol{n} \doteq -r\ \delta\theta\ \hat{\boldsymbol{e}}_r + \delta r\ \hat{\boldsymbol{e}}_\theta\ , \tag{3}$$

$$\boldsymbol{n}\cdot(-r\hat{\boldsymbol{e}}_r) = r|\boldsymbol{n}|\cos(\tau) \implies \frac{1}{r}\frac{\delta r}{\delta\theta} = \pm\lambda, \text{ with } \lambda \doteq \tan(\tau). \tag{4}$$

Hence, the curve's equation is $r = r_o \exp[\pm\lambda(\theta - \theta_o)]$, with $r_o = \rho_1$ and $\theta_o = b_1$. The equation now defines two spirals: a spiral s_1^+ growing counter-clockwise from p_1 (or shrinking clockwise), and a second spiral s_1^- shrinking counter-clockwise from p_1 (or growing clockwise). $\partial\mathcal{W}$ must continue from p_1 in the counter-clockwise direction either "above" s_1^+ or "below" s_1^-; otherwise, the incidence constraint would not have been violated.

Similarly, for the opposite end p_2, let $r_o = \rho_2$ and $\theta_o = a_2$. The solution to equation (4) now defines a spiral s_2^- growing clockwise from p_2 (or shrinking counter-clockwise), and a second spiral s_2^+ shrinking clockwise from p_2 (or growing counter-clockwise). $\partial\mathcal{W}$ must continue from p_2 in the clockwise direction either "above" s_2^- or "below" s_2^+.

Remark 1. $\partial\mathcal{W}$ cannot continue below s_1^- when $\rho_1 \exp(-\lambda\beta) < \rho_2$. In other words, $\partial\mathcal{W}$ cannot continue below s_1^- if this spiral curve cuts l_2 below the point p_2 (Figure 21(a)). To show this, suppose $\partial\mathcal{W}$ continues below s_1^-, which implies that $\partial\mathcal{W}$ bends toward the sensor immediately after r_1. We know that $\partial\mathcal{W}$ does not cross the origin, else nothing is visible under Definition 2.1 and Π would be empty. Hence, $\partial\mathcal{W}$ would have to bend outwards before cutting the ray l_2, otherwise r_2 will be occluded. Since $\partial\mathcal{W}$ is differentiable, there must then be a point p where the normal to $\partial\mathcal{W}$ points towards the origin. Because of Lemma A.1, this point p is not occluded by any other section of $\partial\mathcal{W}$ that is disjointed from r_1 and r_2. Therefore, the vicinity of p is a visible portion of $\partial\mathcal{W}$. This violates our assumption that r_2 succeeds r_1. Thus, when $\rho_1 \exp(-\lambda\beta) < \rho_2$, the first section of the curve f joining r_1 to r_2 coincides with s_1^+.

Remark 2. By symmetry, when $\rho_2 \exp(-\lambda\beta) < \rho_1$ (i.e., s_2^+ cuts l_1 below p_1), the last section of the curve f coincides with s_2^- (which grows clockwise from p_2).

The point p_2 may lie below the intersection of s_1^- with l_2, above the intersection of s_1^+ with l_2, or between both intersections. Likewise, the point p_1 may lie below the intersection of s_2^+ with l_1, above the intersection of s_2^- with l_1, or between both intersections. There are total of 9 combinations of events for case {v,v}, but only 3 of them are independent:

(a) s_1^- cuts l_2 above p_2. Thus, $\rho_1 \exp(-\lambda\beta) > \rho_2$, and this is equivalent to $\rho_2 \exp(\lambda\beta) < \rho_1$. That is, s_2^- cuts l_1 below p_1.

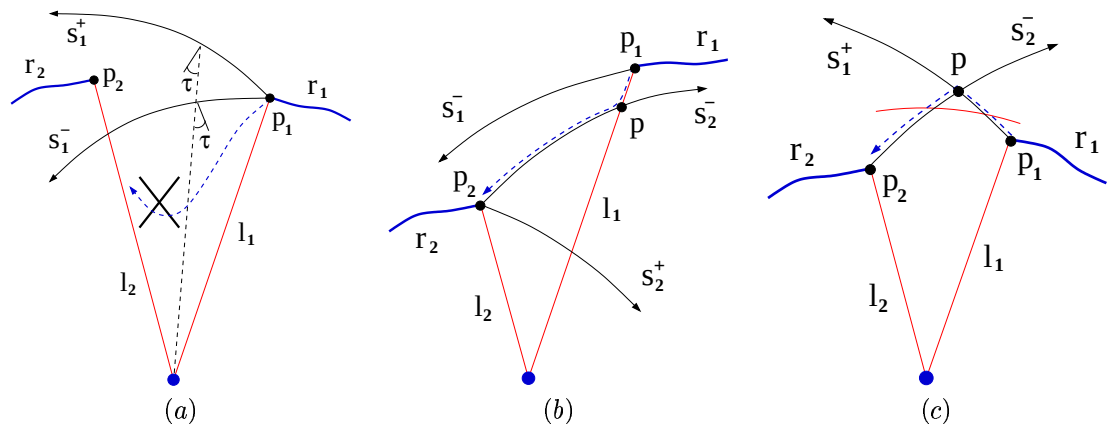


Figure 21: Example of a free-curve construction: (a) this situation is impossible; (b) in this case the free curve is composed of the segment joining p_1 with p and the spiral s_2^- joining p with p_2; (c) here the free curve is composed of the spiral s_1^+ joining p_1 with p and the spiral s_2^- joining p with p_2 (unless p is beyond range, in which case a circular arc of radius r_{max} is added).

(b) s_1^+ cuts l_2 below p_2. Thus, $\rho_1 \exp(\lambda\beta) < \rho_2$, and this is equivalent to $\rho_2 \exp(-\lambda\beta) > \rho_1$. That is, s_2^+ cuts l_1 above p_1.

(c) s_1^- cuts l_2 below p_2 and s_1^+ cuts l_2 above p_2. Thus, $\rho_1 \exp(-\lambda\beta) < \rho_2 < \rho_1 \exp(\lambda\beta)$, and this is equivalent to $\rho_2 \exp(-\lambda\beta) < \rho_1 < \rho_2 \exp(\lambda\beta)$. That is, s_2^+ cuts l_1 below p_1, and s_2^- cuts l_1 above p_1.

Let us analyze the first situation. $\rho_1 \exp(-\lambda\beta) > \rho_2$ is equivalent to $\rho_2 \exp(\lambda\beta) < \rho_1$, which in turn implies that $\rho_2 \exp(-\lambda\beta) < \rho_1$. In other words, both the clockwise-growing s_2^- and the clockwise-shrinking s_2^+ cut l_1 below p_1 (see Figure 21(b)). From Remark 2, the last section of the free curve f coincides with s_2^-. Let p be the intersection between s_2^- and l_1. The free curve f joining r_1 to r_2 is thus composed of the segment joining p_1 with p and the spiral s_2^- joining p with p_2.

A symmetric argument applies to the second situation, when $\rho_2 \exp(-\lambda\beta) > \rho_1$ (i.e., s_2^+ cuts l_1 above p_1), except that Remark 1 is used in this case.

The only remaining situation is (c). Here, $\rho_1 \exp(-\lambda\beta) < \rho_2$ and $\rho_2 \exp(-\lambda\beta) < \rho_1$. From Remarks 1 and 2, these inequalities imply that the first section of f coincides with s_1^+ while the last section of f coincides with s_2^-. Let p be the intersection of s_1^+ and s_2^-. If p is within r_{max}, then the free curve f is composed of the spiral s_1^+ joining p_1 with p and the spiral s_2^- joining p with p_2 (Figure 21(c)). Otherwise p is beyond range, and f is composed of a section of s_1^+, a circular arc of radius r_{max}, and a section of s_2^-.

Case {v,o}: As in the previous case, the curve r_1 was interrupted at $\theta = b_1$ because the incidence constraint was exceeded. The curve r_2, however, was interrupted at $\theta = a_2$ because a portion of $\partial\mathcal{W}$ blocked the sensor's line-of-sight. In order to produce the occlusion, $\partial\mathcal{W}$ must be tangent to l_2 at some point p_t below p_2. We know from Lemma A.1 that the portion of $\partial\mathcal{W}$ producing the occlusion cannot be disjointed from r_1. Thus, p_t is part of the same curve as r_1.

$\partial\mathcal{W}$ cannot continue from r_1 below s_1^-. To show this, suppose $\partial\mathcal{W}$ continues below s_1^-. This implies that $\partial\mathcal{W}$ bends toward the sensor immediately after r_1. But to cause the occlusion, $\partial\mathcal{W}$ has to bend outwards before it reaches the tangent point p_t. Since $\partial\mathcal{W}$ is differentiable, there must be a point where the normal to $\partial\mathcal{W}$ points towards the origin. But we already know that this violates our assumption that r_2 succeeds r_1.

Given that $\partial\mathcal{W}$ cannot continue from r_1 below s_1^-, then $\partial\mathcal{W}$ must continue above s_1^+.

For case {v,o}, it is always true that s_1^+ cuts the ray l_2 below p_2 at some point p. Otherwise, it will be impossible to produce the occlusion at p_t, because $\partial\mathcal{W}$ continues from r_1 above s_1^+. Thus, f is composed of the spiral s_1^+ joining p_1 with p, and the segment joining p with p_2.

Case {v,e}: As before, the incidence constraint was exceeded at $\theta = b_1$. But the curve r_2 was interrupted because the range constraint was exceeded at $\theta = a_2$. That is, $\rho_2 = r_{max}$.

$\rho_1 < r_{max}$ because the point p_1 is within range, which implies that $\rho_1 \exp(-\lambda\beta) < \rho_2$ since $\rho_2 = r_{max}$. This is exactly the situation described in Remark 1 for case {v,v}. Thus, $\partial\mathcal{W}$ cannot continue below s_1^-, and the first section of f coincides with s_1^+.

If s_1^+ cuts the ray l_2 below p_2 at some point p, then f is composed of the spiral s_1^+ joining p_1 with p, and the segment joining p with p_2. Otherwise p is beyond range, and f is composed of a section of s_1^+ and a circular arc of radius r_{max}.

Case {e,e}: This case is trivial. The free curve is a circular arc connecting p_1 with p_2.

Cases {o,o} and {e,o}: These cases are impossible because of Lemma A.1. In case {o,o}, both r_1 and r_2 are occluded. We know from Lemma A.1 that the portion of $\partial\mathcal{W}$ occluding r_2 cannot be disjointed from r_1, and that the portion of $\partial\mathcal{W}$ occluding r_1 cannot be disjointed from r_2. But this situation is impossible.

For the case {e,o}, $\partial\mathcal{W}$ must be tangent to l_2 at some point p_t below p_2. But $\partial\mathcal{W}$ continues after r_1 beyond the maximum range. Therefore, $\partial\mathcal{W}$ has to bend toward the sensor to fall back within range, and then bend outwards before it reaches the tangent point p_t. Again, this violates our assumption that r_2 succeeds r_1, because $\partial\mathcal{W}$ is differentiable and there must be a point where the normal to $\partial\mathcal{W}$ points towards the origin. Hence, this case is impossible.

We have accounted all possible cases. This concludes our proof of Theorem 3.1.