# A Terrain-Covering Algorithm for an AUV*

Susan Hert Sanjay Tiwari Vladimir Lumelsky

Robotics Laboratory
University of Wisconsin–Madison
1513 University Ave.
Madison, WI 53706-1572

**Abstract**

An efficient, on-line terrain-covering algorithm is presented for a robot (AUV) moving in an unknown three-dimensional underwater environment. Such an algorithm is necessary for producing mosaicked images of the ocean floor. The basis of this three-dimensional motion planning algorithm is a new planar algorithm for nonsimply connected areas with boundaries of arbitrary shape. We show that this algorithm generalizes naturally to complex three-dimensional environments in which the terrain to be covered is projectively planar. This planar algorithm represents an improvement over previous algorithms because it results in a shorter path length for the robot and does not assume a polygonal environment. The path length of our algorithm is shown to be linear in the size of the area to be covered; the amount of memory required by the robot to implement the algorithm is linear in the size of the description of the boundary of the area. An example is provided that demonstrates the algorithm's performance in a nonsimply connected, nonplanar environment.

## 1 Introduction

We present an on-line terrain-covering algorithm for an autonomous underwater vehicle (AUV) moving in three dimensions in an unknown underwater environment. The goal is to generate an efficient path for the robot that allows it to cover a given area in its entirety. The on-line nature of the algorithm allows the AUV to explore and provide information about areas of the ocean floor where the terrain is unknown. Such an algorithm may be used, for example, to create a mosaicked image of the ocean floor. This is the context in which we present this algorithm. The algorithm was designed with an emphasis on efficiency: not only does it guarantee that the entire area will be covered, but

*This work was supported in part by Sea Grant Program (National Oceanic and Atmospheric Administration, US Dept. of Commerce) Grant NA46RG048, and the State of Wisconsin.

it ensures that no path segment will be traveled along many times, which is a serious consideration in real-world applications; covering an area multiple times is generally seen as unnecessary and thus a waste of effort.

The basis of our three-dimensional motion planning algorithm is a new planar terrain-covering algorithm. To be able to generalize this planar algorithm directly to the three-dimensional environment, we require that the surface to be covered be such that any vertical line passing through the surface intersects it at exactly one point. This assures that there is a one-to-one correspondence between the points on the surface and those in its projection on the $xy$-plane. Thus, if the planar algorithm generates a path that allows a robot to cover every point in a given planar area, a robot following this path in a three-dimensional environment will also be able to cover every point on a corresponding nonplanar surface. This obviously restricts the types of environments in which our algorithm may be used (*e.g.*, no caves are allowed in the environment).We discuss in Section 7 how this constraint may be relaxed.

The robot's sensors provide it with the information necessary to plan its path and maintain a certain vertical distance from the floor. A system of navigation sensors such as an inertial navigation system, a system of acoustic transponders, or a multisensor integrated navigational system, allows the AUV to localize its position in space relative to a fixed coordinate system [Blidberg, 1995]. From its known position in the environment, the robot must be able to detect and localize points on the ocean floor, compute its distance from the floor and compute the slope of the floor in its vicinity. For this purpose, high-resolution sonar and laser sensing may be used, as described, for example, in the works of Henriksen [Henriksen, 1994], Burke and Rosenstrach [Burke, 1992], Gordon [Gordon, 1992], and Rosenblum and Kamgar-Parsi [Rosenblum, 1992]. Though they must be considered in any implementation of the algorithm, we do not address here the problems that arise due to noisy sensor data.

For producing the mosaicked image of the ocean floor, the robot is equipped with a camera and an internal stabilizer that keeps the camera pointing vertically downward, and the necessary lighting equipment and control procedures [Chu, 1992]. Since a video mosaic provides a better comprehensive picture of the ocean floor, a video or CCD camera is most appropriate. The robot must move over the area of interest in such a way that its camera is able to take a picture of every part of the area. Further, this must be done so there is sufficient overlap in the images to facilitate the creation of the

mosaicked image. Our algorithm is developed with these considerations in mind.

We do not address here the problem of creating the mosaicked image from its components. This is addressed, for example, in [Marks, 1994] and [Haywood, 1986]. Both these works present methods of creating mosaics automatically in real time, which is compatible with the on-line nature of our algorithm. For example, in the work of Marks, *et al.* gaps between images are prevented by observing and evaluating intermediate mosaic results. Since our algorithm is designed to work in real-time, it would allow for such intermediate observation and for the consequent corrections that may be necessary.

While other on-line terrain-covering algorithms for planar environments have been presented in the literature, they either do not generalize well for use in nonpolygonal or three-dimensional environments or are not efficient enough for practical use. Oommen, *et al.* [Oommen, 1987] present an algorithm for covering an unknown planar terrain populated with convex polygonal obstacles; the algorithm is based on an incremental acquisition of the visibility graph of the terrain. Rao and Iyengar [Rao, 1990] extend this method to terrains containing nonconvex polygonal obstacles. Neither of these methods is particularly well suited for use in an underwater environment, where boundaries are anything but polygonal and approximation techniques cannot be reasonably applied. Lumelsky, *et al.* [Lumelsky, 1990] present a planar terrain-covering algorithm for an environment with arbitrarily shaped planar obstacles. Our algorithm, which also allows arbitrarily shaped boundaries, though under a slightly different model, represents an improvement over this method — its upper bound on the path length of the robot is significantly better. Also, unlike the other works, we provide a bound on the amount of memory required by the robot to implement the algorithm. This bound is linear in the size of the boundary description.

The planar terrain-covering algorithm we present may be used for simply and nonsimply connected environments. The simplicity of the algorithm lies in its recursive nature. A robot following this algorithm will zigzag along parallel straight lines to cover a given area. Portions of the area that either would not be covered or would be covered twice using the zigzag procedure are detected by the robot and covered using the same procedure. These smaller areas (*inlets*) may, in turn, contain other areas that are treated in the same way. By covering each inlet as soon as it is detected, they are covered in a depth-first order.

The spacing of the parallel lines in the zigzag pattern is determined by the robot's camera image

width. The shape of the boundary relative to these lines is what gives rise to inlets. Assumptions we make about the sensing capabilities of the robot and the spacing of the parallel lines ensure that every inlet may be detected. The algorithm causes each inlet to be covered exactly once.

The only special procedures necessary for efficiently covering the inlets (which we call *diversion inlets*) that would not be covered or would be covered more than once as the robot zigzags are those for entering and exiting them. The robot enters a diversion inlet by moving along its boundary. After covering a given diversion inlet, the robot exits it by resuming its path of travel as if the diversion inlet did not exist. The algorithm requires that the robot remember the location of every inlet it covers. This assures that after a diversion inlet is detected it will be covered only once. When the area to be covered is not simply connected and contains islands as well as inlets, the same basic procedures are used with only minor modifications to ensure that the area surrounding every island is covered. By remembering certain points along its path, the robot is able to convert the part of the area around each island that would normally not be covered into an artificial inlet. Artificial inlets are covered in the same way that real diversion inlets are covered.

Section 2 presents the assumed models of the robot, the three-dimensional environment and its planar counterpart. Though formulated for a point robot, our algorithm may be easily extended for use by a robot modeled as a circle (in two dimensions) or a sphere (in three dimensions) [Choi, 1995], using the standard technique of growing the boundary of the area by the radius of the robot [Lozano-Pérez, 1983]. The algorithm for a planar region is developed and described in Section 3. Section 4 discusses the generalization of the planar algorithm to the three-dimensional environment. In Section 5, the complexity of the algorithm is presented. The correctness of the algorithm is discussed in Section 6, followed by an example of the algorithm's performance in Section 7. A discussion of the generalization to more complex nonplanar environments is also given in this section.

## 2 The Model

The robot is a point moving in three dimensions. A fixed orthogonal coordinate system $X = (x, y, z)$ is chosen with its origin at the robot's starting point $S$ and $z$ axis passing through the earth's center.

The robot is equipped with sensors and a camera. The sensors allow the robot to determine its own coordinates relative to $X$ and those of any point detected in its sensing region. The *sensing*

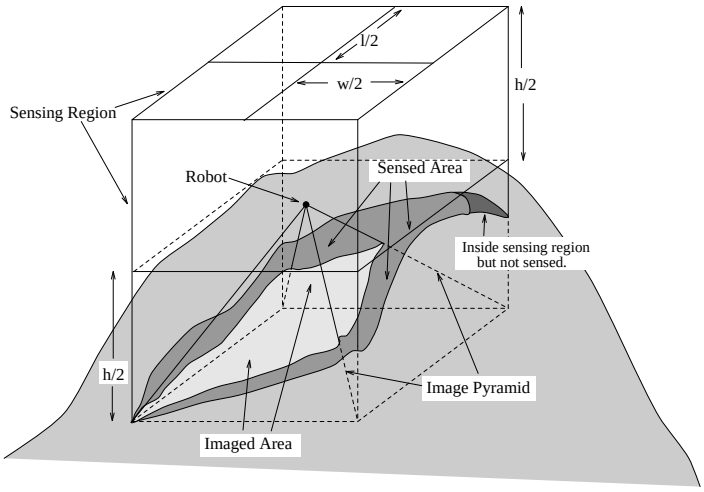



Figure 1: *For a robot at position $R$, its sensing region is a rectangular polyhedron of dimensions $l \times w \times h$ with the robot at its center. The sensed area is portion of the intersection of the sensing region an the ocean floor for which a straight line from $R$ to any point in the region does not intersect the floor at any other point. The imaged area is the intersection of the image pyramid and the sensed area.*

*region* is a rectangular polyhedron of dimensions $l \times w \times h$ ($length \times width \times height$), with the robot at its center. These dimensions are chosen such that, from a vertical distance of $h/2$ from a horizontal plane, the robot's camera will take a picture of a rectangle on the plane of dimensions $l \times w$. The value $h/2$ is determined by the focal length of the robot's camera. The sensors also allow the robot to determine the slope of the floor within its sensing region, which enables it to maintain the vertical distance of $h/2$ from the floor.

The *sensed area* is the portion of the ocean floor that the robot can sense from a particular position $R$. This area is, in general, different from the intersection of the ocean floor with the sensing region. Rather, it consists of all points $p$ on the floor that are in the sensing region and for which a line segment $\overline{pR}$ does not intersect the surface at any other point (Figure 1).

We assume that the robot's camera always points down and the robot continually records the images from the camera as it moves along its path. A surface or area has been *covered* if the robot has taken a picture of every (visible) part of it. The *imaged area* is the portion of the ocean floor of which the robot takes a picture from a particular position $R$. This will be some subset of the sensed area. In particular, it is the portion of the sensed area that lies within the *image pyramid*. This pyramid has its apex at $R$ in the center of the sensing region and base equal to the bottom of the

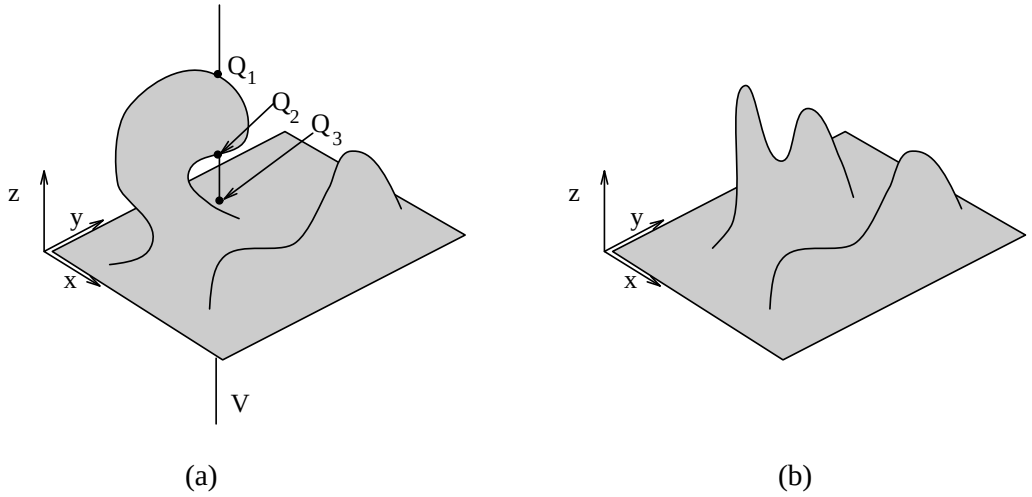



Figure 2: *(a) The area shown is not projectively planar since the vertical line $V$ intersects the surface at three points: $Q_1$, $Q_2$, and $Q_3$. (b) This area is projectively planar — no vertical line will intersect the surface in more than one point.*

sensing region (a rectangle of dimensions $l \times w$) (Figure 1).

The nonplanar surface (area) to be covered, $\mathcal{A}$, is a *vertically projectively planar surface*. That is, a vertical line passing through any point $p$ on the surface $\mathcal{A}$ intersects it at only one point (Figure 2). Other than this, $\mathcal{A}$ may be of arbitrary shape. In particular, it need not be simply connected; it may contain *islands*, or holes, of arbitrary shape (Figure 3). It is assumed that from the robot's starting point $S$ it is able to sense some portion of the area $\mathcal{A}$.

We assume that area $\mathcal{A}$ is bounded between two *threshold surfaces*, $z = z_{\min}$ and $z = z_{\max}$.[1] Portions of the floor below $z_{\min}$ or above $z_{\max}$ are not to be covered (Figure 3). The area is also bounded by a threshold slope, $\mu$. Portions of the floor where the slope is greater than $\mu$ are not to be covered. This threshold slope is inversely proportional to the image width $w$ and is chosen to assure adequate overlap of the images. The range of possible values for $\mu$ is limited by the robot's camera parameters. In particular, if the focal range of the robot's camera is $[a, b]$, then $\mu$ must be chosen to satisfy the relation:

$$\mu < \min(h/3w, (b-a)/(w/2))$$

[1] More generally, these surfaces may be represented as $z = h_{\min}(x, y)$ and $z = h_{\max}(x, y)$, where $h_{\min}$ and $h_{\max}$ are arbitrary functions of $x$ and $y$. For simplicity of explanation, we assume these functions are constant, *i.e.*, the surfaces are horizontal planes.

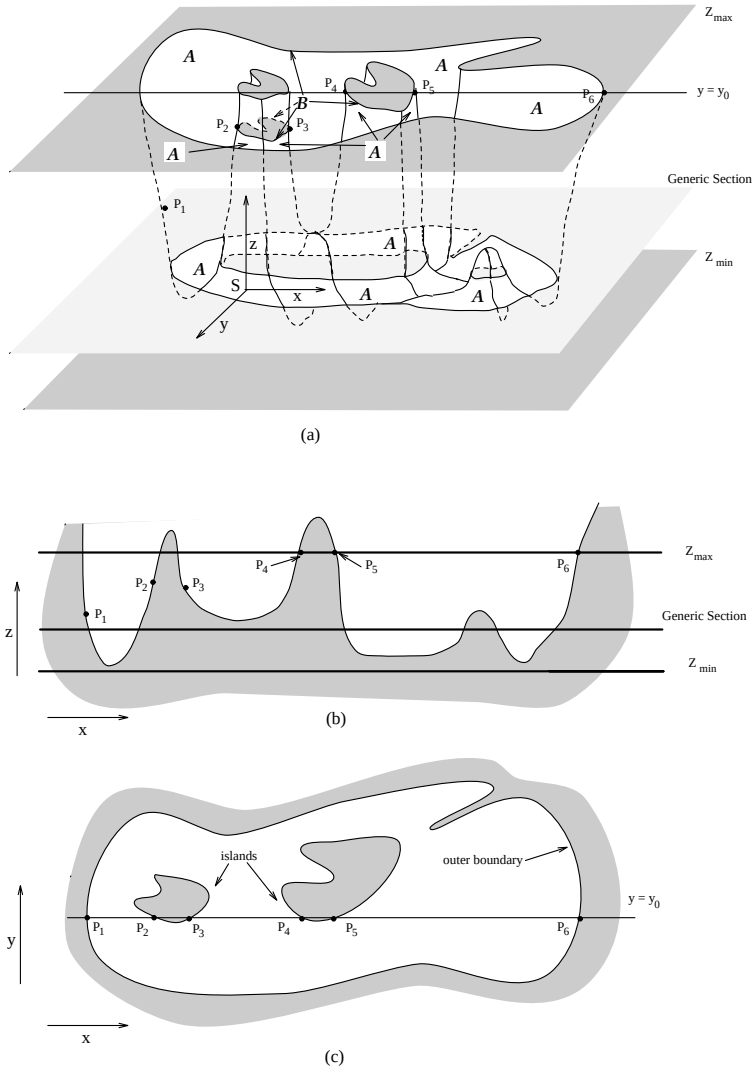



Figure 3: *(a) The robot will cover area $\mathcal{A}$ with boundary $\mathcal{B}$. $\mathcal{B}$ is determined by the intersections of the floor with the planes $z = z_{\min}$ and $z = z_{\max}$ and the curves along the floor where the slope $= \mu$. (b) A vertical cross section in the plane $y = y_0$ of the area of shown in (a). The threshold surfaces $z = z_{\max}$ and $z = z_{\min}$ appear here as line segments. Points $P_1, P_2$, and $P_3$ are points where the slope of the surface exceeds the threshold slope $\mu$. These points belong to the boundary $\mathcal{B}$ of $\mathcal{A}$. (c) Area $\mathcal{A}$ projected onto the $xy$-plane.*

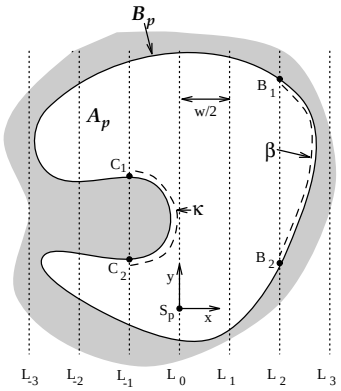



Figure 4: *In the planar area $\mathcal{A}_p$ with planar boundary $\mathcal{B}_p$, the chosen coordinate system at the projected starting point $S_p$ is shown. The grid lines $L_{-3}, \ldots, L_3$ are each at distance $w/2$ from the adjacent lines. The boundary section $\kappa$ between grid lines $L_{-1}$ and $L_0$ with endpoints $C_1$ and $C_2$ is a cape. The boundary section $\beta$ between lines $L_2$ and $L_3$ and between points $B_1$ and $B_2$ is a bay; there are two other unlabeled bays between $L_{-2}$ and $L_{-3}$ with endpoints on grid line $L_{-2}$.*

This assures that images of adjacent portions of the surface will overlap to some extent and every part of the surface that is to be imaged lies within the camera's focal range.

The boundary $\mathcal{B}$ of the area $\mathcal{A}$ consists of a number of simple, nonintersecting closed curves. These are the intersections of the floor with the threshold surfaces and the curves along the floor where the slope is equal to $\mu$ (Figure 3). The robot detects the boundary by analyzing its sensor readings (Section 4.2).

The *planar area* $\mathcal{A}_p$ and *planar boundary* $\mathcal{B}_p$ are defined as the projections of the area $\mathcal{A}$ and boundary $\mathcal{B}$ on the $xy$-plane (*i.e*, plane $z = 0$). The *outer boundary* is the boundary curve that contains all other boundary curves in its interior (Figure 3(c)). A *grid plane* $P_i$ is a vertical plane defined by the equation $x = iw/2$ for some integer $i$ and sensing width $w$. A *grid line* $L_i$ is the intersection of the grid plane $P_i$ and the $xy$-plane (*i.e*, a line in the $xy$-plane defined by the equation $x = iw/2$, for some integer $i$ and sensing width $w$). This definition assures that, from a position on grid line $L_i$, the robot is able to sense everything between the two flanking grid lines $L_{i-1}$ and $L_{i+1}$ (Figure 4). It is assumed that every point on $B_p$ may be sensed from at least one grid line and that the area $\mathcal{A}_p$ is connected. Further, we assume that any straight line intersects $\mathcal{B}_p$ in at most a finite number of points. That the robot is able to detect a portion of $\mathcal{A}$ from the starting point $S$ implies that the projected starting point, $S_p$, lies in the interior of $\mathcal{A}_p$.

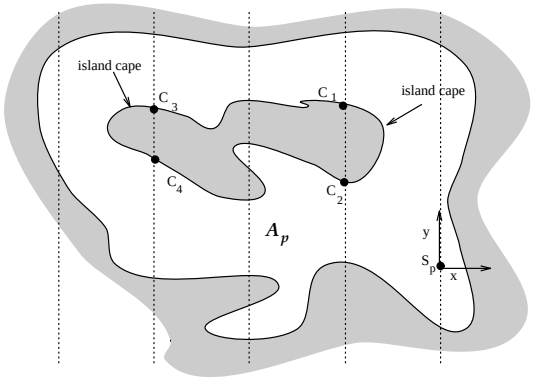
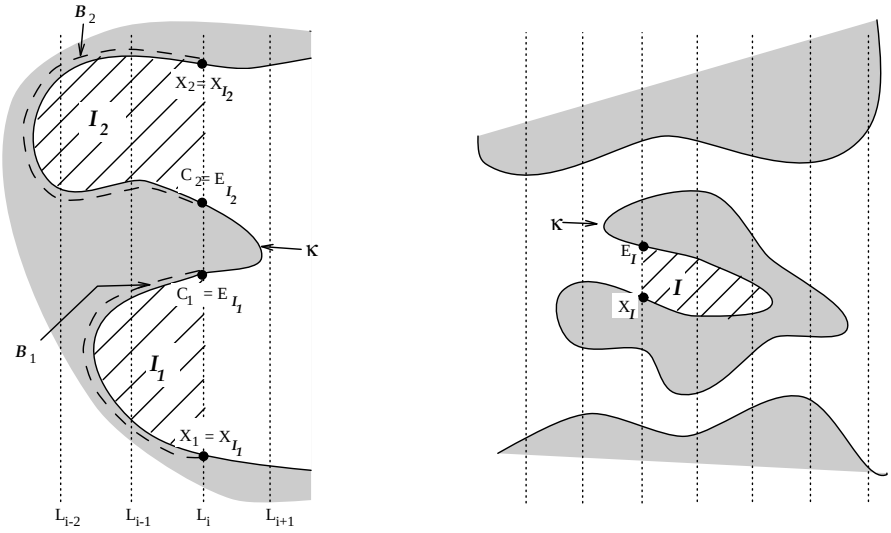



Figure 5: *The two island capes are* $(C_1, C_2)$ *and* $(C_3, C_4)$. *The island cape points are* $C_1, C_2, C_3$, *and* $C_4$.

Certain sections of the boundary $\mathcal{B}$, called *capes* and *bays*, are of interest in characterizing its shape relative to the chosen coordinate system. These terms are defined using the planar area $\mathcal{A}_p$ and its boundary $\mathcal{B}_p$. When viewed from the interior of $\mathcal{A}_p$, capes appear as convexities and bays as concavities. More precisely, a cape $\kappa = (C_1, C_2)$ is a continuous portion of $\mathcal{B}_p$ lying between two consecutive grid lines, with endpoints $C_1$ and $C_2$ on the same grid line, such that points on the line segment $\overline{C_1C_2}$ in the neighborhood of each endpoint lie in the exterior of $\mathcal{A}_p$ (Figure 4). The endpoints $C_1$ and $C_2$ of a cape $\kappa$ are referred to as *cape points*. A bay $\beta = (B_1, B_2)$ is defined similarly, with the difference that points in the neighborhood of the endpoints of $\overline{B_1B_2}$ lie in the interior of $\mathcal{A}_p$. The points $B_1$ and $B_2$ are referred to as *bay points*. Note that the outer boundary of every area contains at least two bays. Also, the boundary of every island that crosses a grid line contains at least two capes, one at each horizontal extreme. These capes are called *island capes*. The four endpoints of the island capes are *island cape points* (Figure 5).

Every cape that is not an island cape gives rise to one or two *inlets*. An *inlet* is a portion of the area $\mathcal{A}_p$ bounded by a grid line segment and an *inlet boundary*. In general, the inlet boundaries associated with a cape $\kappa = (C_1, C_2)$ are defined as follows. Assume the cape points $C_1$ and $C_2$ lie on grid line $L_i$. Let $X_1$ and $X_2$ be the two intersections of $L_i$ and the boundary curve containing $\kappa$ on either side of the cape points, with $X_1$ closer to $C_1$ than to $C_2$ (Figure 6). For the cape $\kappa$, the inlets' boundaries are the boundary sections $\mathcal{B}_1 = (C_1, X_1)$ and $\mathcal{B}_2 = (C_2, X_2)$ that do not include the cape $\kappa$. Note that $B_1$ and $B_2$ may cross any number of grid lines and may contain the boundaries of




(a) (b)

Figure 6: *(a) For cape $\kappa$, there are two inlets $\mathcal{I}_1$ and $\mathcal{I}_2$, with boundaries $\mathcal{B}_1$ and $\mathcal{B}_2$. The cape points $C_1$ and $C_2$ are the entrance and exit points of inlets $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. Points $X_1$ and $X_2$ are the inlets' exit points. The line segments $\overline{E_{\mathcal{I}_1} X_{\mathcal{I}_1}}$ and $\overline{E_{\mathcal{I}_2} X_{\mathcal{I}_2}}$ are the inlet doorways. (b) For cape $\kappa$ on the island, there is only one associated inlet, $\mathcal{I}$.*

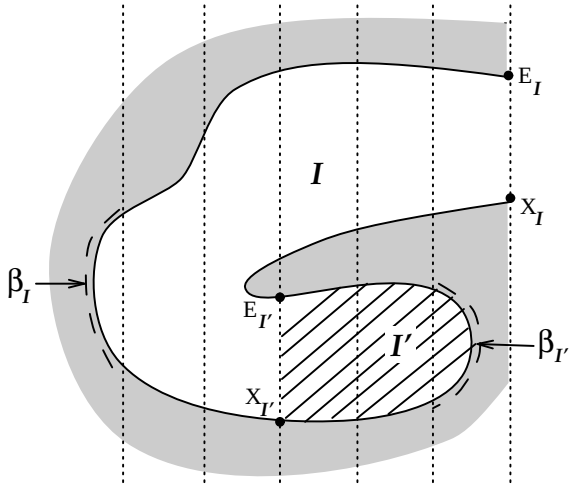
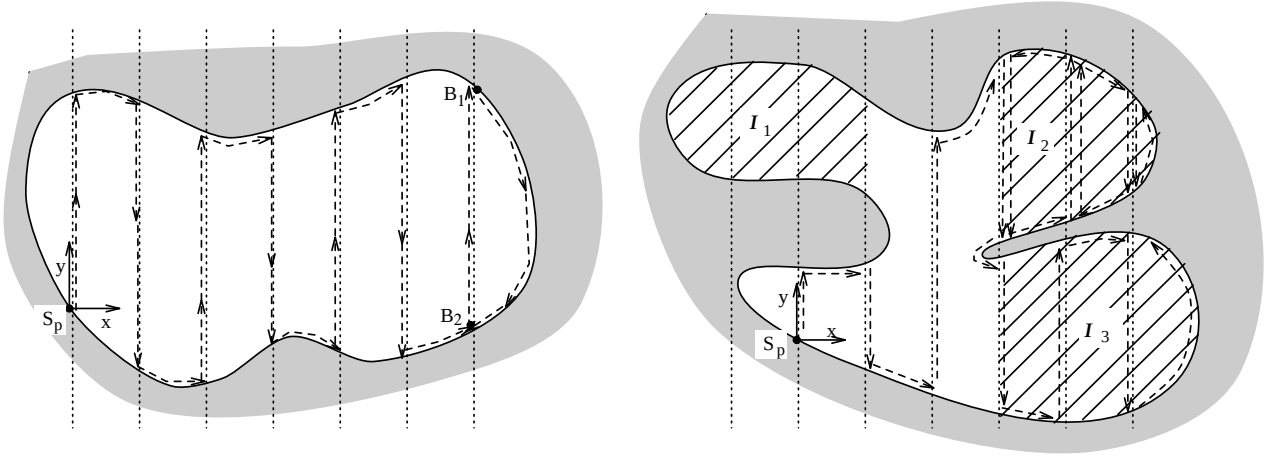



Figure 7: *The boundary of inlet $\mathcal{I}$ ($E_{\mathcal{I}} X_{\mathcal{I}}$) contains two bays, $\beta_{\mathcal{I}}$ and $\beta_{\mathcal{I}'}$. Bay $\beta_{\mathcal{I}}$ is the inlet bay for $\mathcal{I}$ since $\beta_{\mathcal{I}'}$ is on the boundary of inlet $\mathcal{I}'$, contained in $\mathcal{I}$. Bay $\beta_{\mathcal{I}'}$ is the inlet bay for $\mathcal{I}'$.*




(a) (b)

Figure 8: *(a) By starting at point $S_p$ and simply zigzagging along the grid lines, the robot produces the path shown here (dashed line) that covers the entire area. (b) When there are inlets, the situation is more complex. When the robot simply zigzags, inlet $\mathcal{I}_1$ is never covered. Inlet $\mathcal{I}_2$ must be covered twice in order for the robot to reach inlet $\mathcal{I}_3$.*

other inlets (*subinlets*) (Figure 7). Note also that capes on island boundaries may give rise to only a single inlet (Figure 6(b)). In each case, $C_j$ is the *inlet entrance point* and $X_j$ is the *inlet exit point.* For inlet $\mathcal{I}$ these are denoted $E_{\mathcal{I}}$ and $X_{\mathcal{I}}$, respectively. The line segment $\overline{E_{\mathcal{I}}X_{\mathcal{I}}}$ is the *inlet doorway.* The robot may *lock* an inlet doorway by remembering its endpoints. A doorway is *unlocked* when the robot removes the endpoints of this line segment from its memory. Associated with each inlet $\mathcal{I}$ is its *inlet bay*, $\beta_{\mathcal{I}}$. Each inlet's boundary contains at least one bay; if $\mathcal{I}$ contains subinlets, its boundary will contain more than one bay. Generally, bay $\beta_{\mathcal{I}}$ is the bay on the boundary of inlet $\mathcal{I}$ that is not part of any subinlet's boundary (Figure 7).

Given this model, the task is to generate a path for a robot starting at point $S$ that causes it to cover the entire area $\mathcal{A}$ in an efficient manner. We first develop in Section 3 a terrain-covering algorithm for a planar area $\mathcal{A}_p$ with boundary $\mathcal{B}_p$. This is then generalized in Section 4 to the projectively planar surface $\mathcal{A}$ and boundary $\mathcal{B}$.

# 3 The Planar Algorithm

Key to our terrain-covering algorithm is a simple zigzagging pattern of motion in which the robot moves back and forth along successive grid lines, sweeping across an area from either left to right or right to left. Figure 8(a) shows the path of a robot moving in this fashion in a simply connected area, the boundary of which contains only two bays and no capes. By starting at one of the bay points on the boundary $\mathcal{B}_p$ and zigzagging until it encounters the other bay, the robot is able to cover the entire area.

However, as Figure 8(b) illustrates, this simple zigzagging motion will not suffice to cover areas with boundaries that contain capes, nor will it work when the area is not simply connected or when the starting point is chosen arbitrarily. The inlet labeled $\mathcal{I}_1$ in Figure 8(b) is never encountered by the robot as it zigzags along the grid lines, and thus remains uncovered. In contrast, the robot must retravel its path in inlet $\mathcal{I}_2$ in order to reach the inlet $I_3$. Inlets such as $\mathcal{I}_1$ and $\mathcal{I}_2$ are referred to as *diversion inlets*, or simply *D-inlets*, since they require that the robot divert from its normal zigzagging motion in order to cover them efficiently.

Given the assumed range of the robot's sensors relative to the spacing of the grid lines, it will be able to detect the presence of all cape points, and thus the presence of all inlets that it will not cover. Therefore, one simple way to cause the entire area to be covered is to have the robot remember the locations of the inlets it notices but does not cover and, after covering the rest of the area $\mathcal{A}_p$, return to these inlets and cover them. However, this backtracking is both undesirable and unnecessary. Since the robot can notice each inlet as it is moving, it can immediately make a diversion from its path to cover this inlet and then return to the point at which the diversion was made and continue on its merry way. In this way, the entire area will be covered without any extensive backtracking. This is the strategy of the algorithm presented here. Further, the procedures presented assure that every inlet is covered only once.

By not assuming a polygonal environment for our algorithm, we have differentiated this work from most on-line terrain-covering algorithms presented in the literature. The exception is the Seed Spreader algorithm presented in [Lumelsky, 1990], which also assumes arbitrarily shaped boundaries. The Seed Spreader algorithm also dictates that a robot zigzag along grid lines to cover a given area, making diversions to cover obstacles (islands and inlets) it encounters along the way. The algorithm we present here differs from the Seed Spreader algorithm in the following significant ways. It is

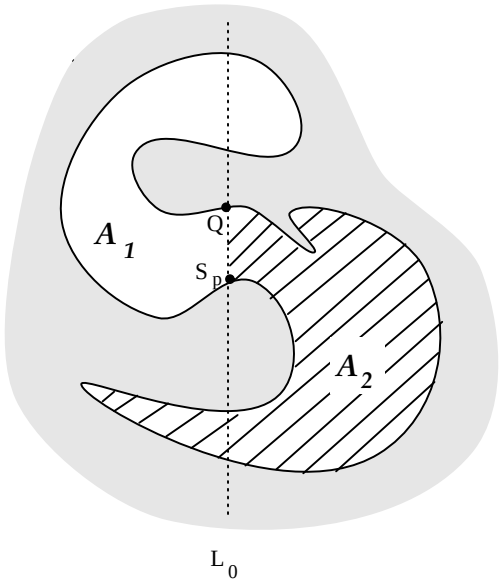



Figure 9: *The area $\mathcal{A}_p$ is divided into two smaller areas $\mathcal{A}_1$ and $\mathcal{A}_2$ by the placement of the point $S_p$. The line segment $\overline{QS_p}$ is the boundary between $\mathcal{A}_1$ and $\mathcal{A}_2$.*

assumed in [Lumelsky, 1990] that there is a known rectangular boundary around the area to be covered. Obstacles may cross this boundary, thus making it irregularly shaped. However, there is a distinction made between the outer, polygonal boundary and the boundaries of the nonpolygonal obstacles; they are treated differently in the algorithm with the result that the outer boundary is traveled along at most once and the boundaries of obstacles (especially those that cross many grid lines) may be traveled along many times. We make no such assumption or distinction in this algorithm, which results in a statement of the algorithm with fewer special cases and a shorter path length for the robot in the worst case. It should be noted that the Seed Spreader algorithm is designed to handle environments in which there are portions of the boundary that are not visible from any grid line segment. The algorithm presented here assumes there are no such boundary sections. In this sense, the Seed Spreader algorithm may be said to be more general than our algorithm. However, this algorithm is not as easily implementable since it is not explicitly stated in [Lumelsky, 1990] what points must be remembered by the robot for proper execution of the algorithm or how obstacles or occlusions are detected by the robot. These are issues we address in this paper. The detection problems become more complicated when occluded boundary sections are allowed and have thus not yet been addressed.

Since we make no assumptions about the location of the starting point of the robot relative to the boundary of the area $\mathcal{A}_p$ (other than that it is somewhere in the interior of $\mathcal{A}_p$), the starting point may be seen to divide $\mathcal{A}_p$ into two areas $\mathcal{A}_1$ and $\mathcal{A}_2$, as shown in the example in Figure 9. The boundary between $\mathcal{A}_1$ and $\mathcal{A}_2$ is dependent upon the placement of $S_p$ and the shape of the boundary. It represents the limit of the area covered by the robot when it moves in either direction along the grid line $L_0$ away from $S_p$ and follows the algorithm presented here. In general, this boundary consists of a number of grid line segments and boundary segments. It arises as an artifact of the algorithm and is never calculated by the robot. Since the robot uses the same algorithm to cover $\mathcal{A}_1$ and $\mathcal{A}_2$, it is guaranteed that the entire area $\mathcal{A}_p$ may be covered by a robot starting at $S_p$ using the following three steps:

1. Cover the area $\mathcal{A}_1$;
2. Return to $S_p$;
3. Cover the area $\mathcal{A}_2$.

The following sections describe the procedures required for these steps. Sections 3.1 and 3.2 explain, respectively, the procedures for covering simply connected and nonsimply connected areas. Section 3.3 explains the procedure for returning to the starting point $S_p$.

### 3.1 Covering a Simply Connected Area

To guarantee that a given area will be covered in its entirety, procedures are needed for detecting and covering inlets that arise due to capes along the boundary. The robot must also be able to distinguish between the inlets that it would normally cover only once and those that it either would not cover or would cover twice (*i.e.*, between nondiversion and diversion inlets). Since parts of the area that are not inlets will be covered as the robot simply zigzags along the grid lines, these procedures are sufficient to guarantee that the entire area is covered. Here we present such procedures.

Diversion inlets are covered as they are encountered, in a last-in-first-out manner, following the same procedures for each. Though these inlets could be covered by a robot zigzagging from the doorway toward the inlet bay, we choose to have the robot first move along the boundary of the inlet until it encounters the inlet bay and then zigzag back toward the doorway. This makes the procedures generalize easily for covering nonsimply connected areas.

The procedures presented here assure that each inlet is covered only once and that, after a D-inlet has been covered, the robot will continue on as if it had not encountered the inlet. It locks every D-inlet after covering it and, upon encountering the inlet doorway again, treats the locked doorway as if it were a piece of the boundary and not a doorway. In other words, it does not reenter the inlet.

The boundaries of each of areas $\mathcal{A}_1$ and $\mathcal{A}_2$ contain exactly one more bay than cape (Section 6). Every bay for which there is an associated cape is part of a D-inlet boundary; the remaining bay is not. Traveling along a bay causes the robot to reverse its $x$ direction of motion. Inside a D-inlet, this is an indication that the robot should begin zigzagging back toward the doorway; outside a D-inlet, this is an indication that the robot should stop zigzagging since the entire area will have been covered.

The main procedure of our algorithm, *CoverArea*, which causes the robot to zigzag along the grid lines and cover D-inlets as it encounters them, is shown in Figure 10. The steps in this procedure for detecting, covering, and exiting D-inlets are explained in the following sections. Section 3.1.1 describes the means of detecting a D-inlet. The procedure for covering a D-inlet, *CoverDInlet*, is shown in Figure 15 and described in Section 3.1.2. The means of detecting that the robot has traveled along a bay or encountered a D-inlet doorway are described in Sections 3.1.3 and 3.1.4, respectively. Section 3.1.5 describes the procedure *ExitDInlet*, which is shown in Figure 18. (The call to the procedure *ExitDInlet* and the other steps for exiting an inlet appear in the procedure *CoverArea* since *CoverDInlet* calls *CoverArea*.)

### 3.1.1 Detecting a D-inlet Entrance Point

Since every inlet arises due to a cape, the robot detects a D-inlet entrance point by noticing a cape and deciding, based on the manner in which it is moving at the time, which cape point is the entrance point of the inlet it would not cover or would cover twice. A cape may be detected in one of two ways: the robot will either move along the cape while following the boundary or it will notice the cape points in its sensing region.

While moving along the boundary, the robot may detect that it has traveled along a cape as follows. Assume that, while traveling along grid line $L_i$, the robot encounters the boundary at a point $C_1$. If it then moves along the boundary to a point $C_2$, which is also on $L_i$, and any movement along $L_i$ from $C_2$ toward $C_1$ would cause the robot to move into the exterior of $\mathcal{A}_p$, then it has

**Input:**
$MoveDir$ : direction to move along current grid line,
either $y^+$ or $y^-$, for the positive or negative $y$ direction
$TurnDir$ : direction to turn when boundary is encountered, either $Left$ or $Right$
$i$ : the current grid line index
$\mathcal{I}$ : inlet robot is currently inside (NULL if robot is not inside an inlet)

**procedure** $CoverArea$
{
$Done \leftarrow$ False;
while not $Done$ do
{
do
Move along $L_i$ in direction $MoveDir$;
if a D-inlet entrance point is detected then $CoverDInlet()$;
until boundary is hit or $S_p$ is reached;
if robot at $S_p$ then
$Done \leftarrow$ True;
else
Turn in direction $TurnDir$;
do
Move along boundary;
if a D-inlet entrance point is detected then $CoverDInlet()$;
until reach $L_j$ for some $j$;
if at the doorway of inlet $\mathcal{I}$ then
$ExitDInlet(\mathcal{I})$;
Lock doorway of inlet $\mathcal{I}$;
Unlock doorways of inlets contained in $\mathcal{I}$;
$Done \leftarrow$ True;
else if the robot moved along a bay then
$Done \leftarrow$ True;
else
$i \leftarrow j$;
$MoveDir \leftarrow Opposite(MoveDir)$;
$TurnDir \leftarrow Opposite(TurnDir)$;
}
}

Figure 10: *The procedure for covering an area.*

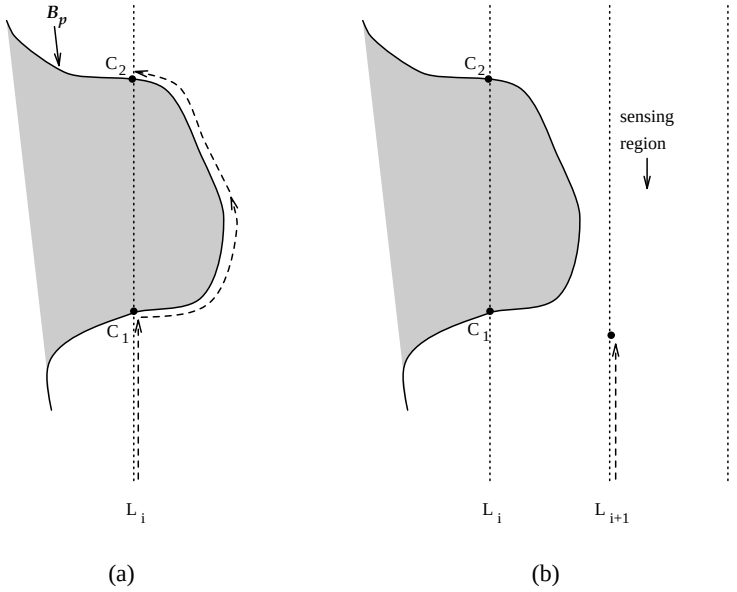



Figure 11: *(a) When the robot arrives at the point $C_2$ on $L_i$ and cannot move toward $C_1$ along $L_i$ without crossing the boundary $B_p$, this means that it has traveled along a cape. (b) From its position on grid line $L_{i+1}$, the robot is able to sense the cape point $C_1$ on grid line $L_i$ as the endpoint of a boundary section on the left side of its sensing region.*

moved along a cape; $C_1$ and $C_2$ are cape points, and $C_2$ is a D-inlet entrance point (Figure 11(a)). Note that the impossibility of the movement from $C_2$ toward $C_1$ in the interior of $\mathcal{A}_p$ requires no movement by the robot; it can be determined with the robot's sensors.

Cape points may be detected by the robot's sensors as the endpoints of boundary segments in the sensing region. These endpoints appear on either the left or right side of the region (Figure 11(b)). When the first cape point is sensed, the robot remembers it but takes no action until the second cape point is sensed. At that time, the robot decides which of the cape points is the D-inlet entrance point $E_{\mathcal{I}}$, depending on its direction of motion and whether it is already inside a D-inlet or not.

When the robot is not inside a D-inlet, every cape it detects within its sensing region gives rise to an inlet that it would not encounter by simply zigzagging along the grid lines. That is, one of the cape points would not be visited. This cape point is the D-inlet entrance point $E_{\mathcal{I}}$. Specifically, assume that the robot will move from grid line $L_i$ to $L_{i+1}$. If the cape lies between $L_i$ and $L_{i+1}$ with cape points on $L_{i+1}$ then $E_{\mathcal{I}}$ is the first cape point detected. If the cape lies between $L_i$ and $L_{i-1}$ with cape points on $L_{i-1}$, $E_{\mathcal{I}}$ is the second cape point detected (Figure 12).

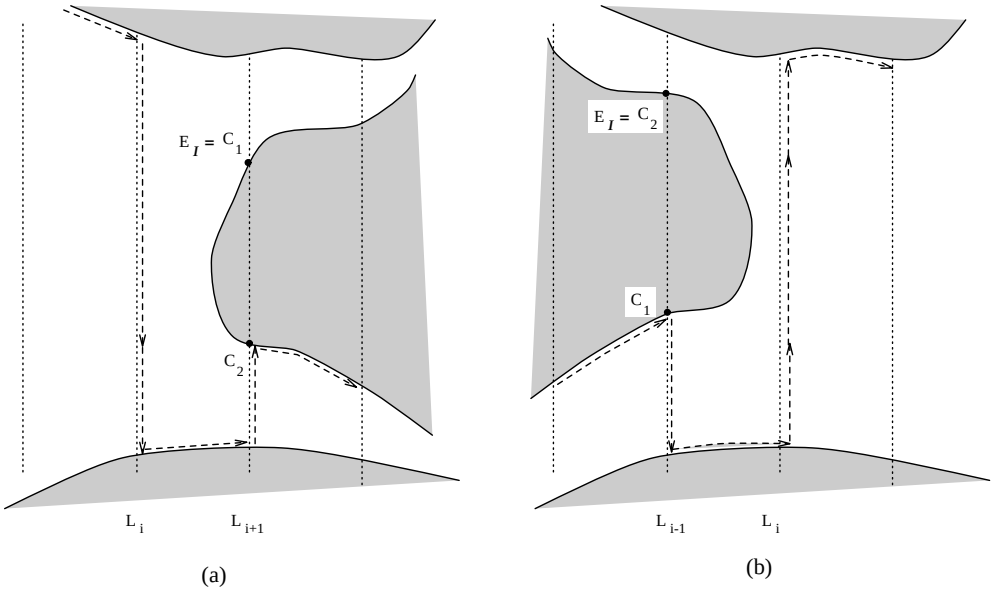



Figure 12: *While not inside another D-inlet and moving from $L_i$ to $L_{i+1}$, the robot detects a cape. The path shown is the one the robot would follow without the diversion to cover the inlet. (a) If the cape points are on grid line $L_{i+1}$, cape point $C_1$ would not normally be encountered by the robot. Thus $E_{\mathcal{I}} = C_1$. (b) If the cape points are on grid line $L_{i-1}$, cape point $C_1$ will have already been visited and $C_2$ will be missed. In this case, $E_{\mathcal{I}} = C_2$.*

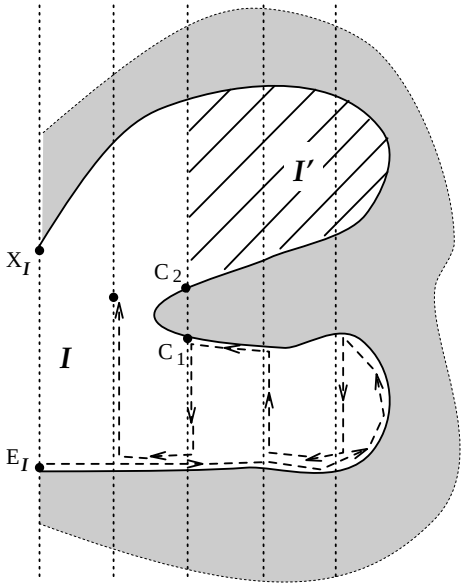



Figure 13: *While covering the inlet $\mathcal{I}$, the robot detects the cape $(C_1C_2)$ with endpoints on the previous grid line. Cape point $C_1$ has already been visited. Thus, $C_2$ is the entrance point for D-inlet $\mathcal{I}'$.*

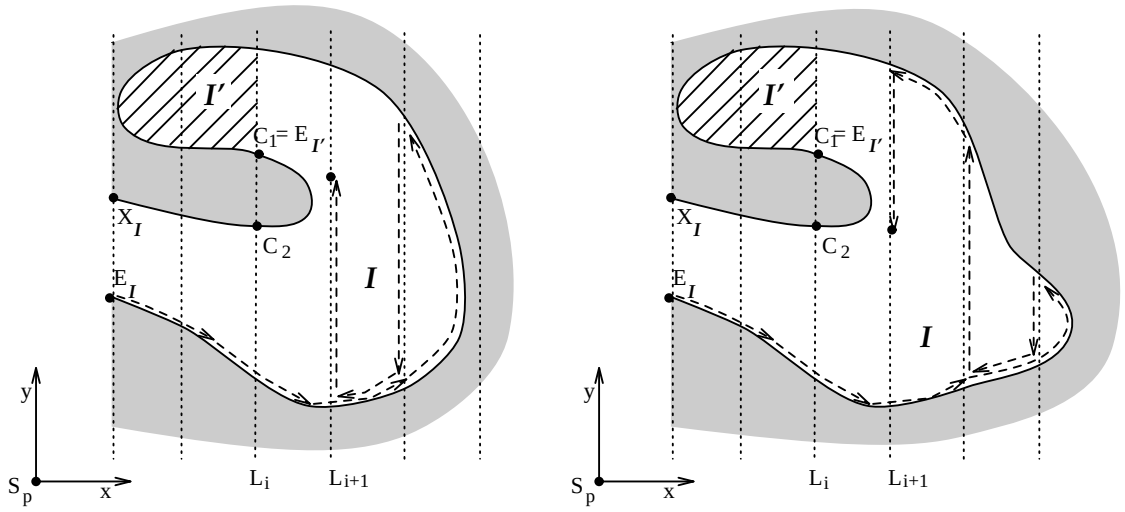



Figure 14: *A D-inlet $\mathcal{I}$ that contains another D-inlet $\mathcal{I}'$ is shown. In each case, if the robot moved along the boundary of inlet $\mathcal{I}$ from $E_{\mathcal{I}}$ to $X_{\mathcal{I}}$, it would move in the counterclockwise direction and encounter cape point $C_1$ before $C_2$. Therefore $C_1 = E_{\mathcal{I}'}$ (a) The robot is moving in the $y^+$ direction along $L_{i+1}$ when the cape points are detected; $C_1$ is the second cape point sensed. (b) The robot is moving in the $y^-$ direction along $L_{i+1}$ when the cape points are detected; $C_1$ is the first cape point sensed.*

When the robot is inside a D-inlet $\mathcal{I}$ and detects cape points within its sensing region, it will generally be traveling from the inlet bay back toward the inlet doorway. Therefore, a pair of cape points that appear on the previous grid line indicates the presence of a D-inlet $\mathcal{I}'$ that would not normally be covered. In this case, the entrance point for the subinlet $\mathcal{I}'$ is the cape point that the robot has not already visited. This is the second cape point sensed (Figure 13). For cape points that appear on the next grid line, we designate that $E_{\mathcal{I}'}$ is the cape point that the robot would encounter first if it were simply following the boundary of the containing inlet $\mathcal{I}$ from $E_{\mathcal{I}}$ to $X_{\mathcal{I}}$. This ensures that the robot will not reach the doorway of inlet $\mathcal{I}$ before covering all its subinlets. To determine which cape point is the D-inlet entrance point in this case, it is necessary for the robot to determine the general direction of motion (clockwise or counterclockwise) along the inlet boundary from $E_{\mathcal{I}}$ to $X_{\mathcal{I}}$. This is easily determined when the robot enters the inlet $\mathcal{I}$. If the direction is counterclockwise, as in Figure 14, then if the robot is moving in the positive $y$ direction ($y^+$), the entrance point is the second cape point detected; if the robot is moving in the negative $y$ direction ($y^-$), the entrance point is the first cape point detected. When the robot is moving clockwise from $E_{\mathcal{I}}$ to $X_{\mathcal{I}}$, the situations are reversed: If the robot is moving in the $y^+$ direction, the entrance point is the first cape point; for motion in the $y^-$ direction, the entrance point is the second point detected.

### 3.1.2 Covering an Inlet

When a D-inlet entrance point $E_{\mathcal{I}}$ that is not part of a locked doorway is detected, the robot remembers $E_{\mathcal{I}}$ (so it can know when it has returned to the doorway of this inlet (Section 3.1.4), immediately moves to it (if it is not already there), and begins to cover the inlet. D-inlets are covered using the *CoverArea* procedure, after the robot has moved along the boundary of the inlet to the inlet bay (Section 3.1.3). It covers any other D-inlets it encounters in the process in the same way, in a last-in-first-out order. The procedure for covering D-inlets is shown in Figure 15.

### 3.1.3 Detecting a Bay

To recognize that an entire area has been covered and to properly cover any D-inlet, the robot must be able to detect when it has traveled along a bay on the boundary. This requires that the robot remember at all times the last encountered point of intersection of a grid line and the boundary. Let $L_i$ be the grid line and $Q_1$ the point of intersection. If the robot follows the boundary and arrives at

```
Input:
    I: the inlet to be covered;

procedure CoverDInlet
{
    Move to E_I;
    Store E_I;
    do
        Move along boundary;
        if a D-inlet entrance point is detected then
            CoverDInlet();
    until robot has moved along a bay;
    CoverArea();
}
```

Figure 15: *The procedure for entering and covering a D-inlet in a simply connected area.*

a second point $Q_2$ on $L_i$ without crossing any other grid lines, and if it is possible to move from $Q_2$ toward $Q_1$ along $L_i$ and still remain in the interior of $\mathcal{A}_p$, then the boundary section traveled along from $Q_1$ to $Q_2$ is a bay. Note that it is not necessary for the robot actually to move toward $Q_1$; the possibility of this motion can be detected using its sensors.

### 3.1.4 Detecting a D-Inlet Doorway

After covering a D-inlet $\mathcal{I}$, the robot must know when it has arrived back at the inlet's doorway. It may arrive there at either endpoint, $E_\mathcal{I}$ or $X_\mathcal{I}$. When it enters the inlet, the robot stores $E_\mathcal{I}$ in its memory and can thus easily recognize if it revisits this point. To detect that it has arrived at the exit point $X_\mathcal{I}$, the robot must simply recognize that it has exited all D-inlets encountered after entering inlet $\mathcal{I}$ and has followed the boundary to a point on the entrance point's grid line (Figure 16).

### 3.1.5 Exiting a D-Inlet

Upon reaching a D-inlet doorway, either at the entrance point $E_\mathcal{I}$ or at the exit point $X_\mathcal{I}$, to ensure that the doorway has been covered, the robot moves along it to the other endpoint. Then the robot must resume moving as if it had never entered the D-inlet. If the robot moved along a cape to $E_\mathcal{I}$, then, after covering the inlet doorway, it simply continues on as if it had not yet reached a grid line.

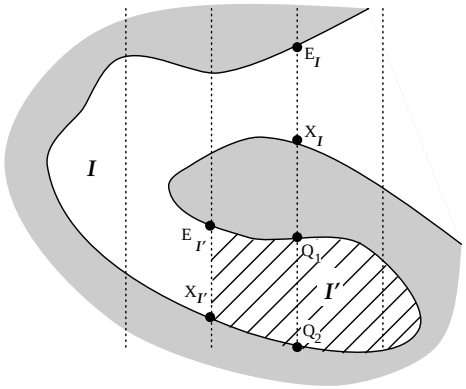
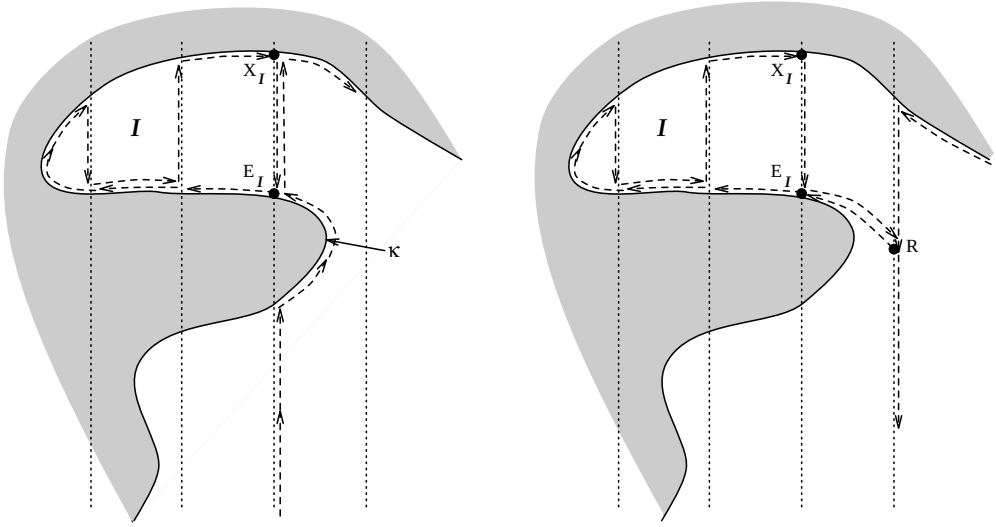



Figure 16: *When the robot arrives at each of the points $Q_1$ and $Q_2$, which are on the same grid line as $E_{\mathcal{I}}$, it will know it is not at the doorway of inlet $\mathcal{I}$ since these points are on the boundary of subinlet $\mathcal{I}'$.*




(a) (b)

Figure 17: *(a) The robot moved along cape $\kappa$ to enter D-inlet $\mathcal{I}$. When it arrives at $X_{\mathcal{I}}$, it moves along the doorway to $E_{\mathcal{I}}$ and then continues as if it had not entered $\mathcal{I}$, by moving back to $X_{\mathcal{I}}$ and along the boundary toward $L_{i+1}$. (b) At point $R$, the robot detects the D-inlet entrance point $E_{\mathcal{I}}$ and moves to it. Upon arrival at point $X_{\mathcal{I}}$, the robot moves along the doorway to $E_{\mathcal{I}}$ and from there back to $R$.*

**Input:**
$\mathcal{I}$: the inlet the robot is exiting

**procedure** *ExitDInlet*
{
Move along doorway;
if the robot did not move along a cape to enter $\mathcal{I}$ then
Move to $E_{\mathcal{I}}$;
Move back to point from which $E_{\mathcal{I}}$ was detected;
}

Figure 18: *The procedure for exiting an inlet.*

That is, it continues to follow the boundary, designating the doorway of the inlet as a portion of the boundary (Figure 17(a)). If the robot entered the D-inlet because it detected $E_{\mathcal{I}}$ with its sensors, then when exiting it moves to $E_{\mathcal{I}}$ after covering the doorway, and from there back to the point at which it detected $E_{\mathcal{I}}$ (Figure 17(b)). This exiting procedure is summarized in Figure 18.

When the robot exits a D-inlet $\mathcal{I}$, it locks the doorway to prevent itself from reentering this inlet. The locked doorways of the subinlets within $\mathcal{I}$ can be unlocked since no locked inlets are ever reentered and thus the subinlets will never be encountered again.

### 3.2 Covering a Nonsimply Connected Area

When the area $\mathcal{A}_p$ contains islands that are not completely contained between successive grid lines, the procedures outlined in Section 3.1 are not sufficient. To see why, note that, unlike the capes on other portions of the boundary, island capes have no associated bays (just as the outer boundary has two bays with no associated capes) (Figure 5). When the robot detects one of the island capes while following the procedures in Section 3.1, it will move to one of the island cape points as if it has encountered a D-inlet. It will then follow the boundary of the island in search of a bay, covering other D-inlets as it encounters them. However, since neither island cape has a bay associated with it, the robot will never stop following the island boundary. For example, in Figure 5, if the robot starts at the point $S_p$ and moves in the $y^+$ direction, it will detect the island cape $(C_1, C_2)$ and move to $C_1$ as if it were entering a D-inlet. It will then follow the boundary indefinitely. What is needed, then, is the ability to recognize islands and to cover the area around them efficiently

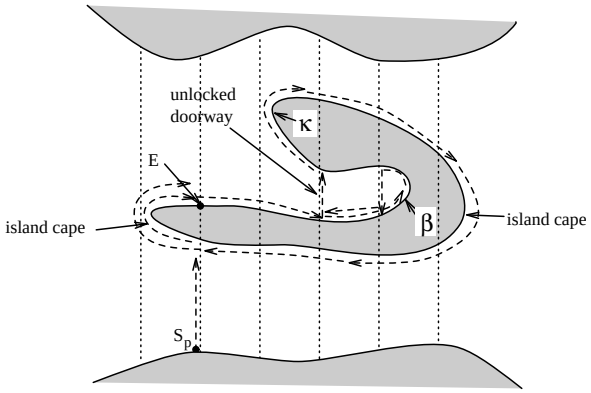



Figure 19: *The robot moves along the island cape to point $E$ and begins to follow the boundary as if entering a D-inlet. After moving along bay $\beta$, the robot assumes it has encountered the inlet bay associated with the entrance point $E$. It zigzags along the grid lines until it travels along the cape $\kappa$, and begins to follow the island boundary again. When the robot arrives back at point $E$, it will have traveled along three capes and only one bay. It may therefore conclude that it has encountered an island. At this point, it recognizes that $\beta$ is actually the inlet bay associated with cape $\kappa$, the next cape traveled along after $\beta$. However, the doorway to this inlet is still unlocked, so the robot locks it and then proceeds to cover the area around the island.*

The procedures presented in this section allow the robot to detect islands and to make note of the island cape points so it may properly cover the area surrounding the islands.

### 3.2.1 Detecting an Island

The robot can detect that it has encountered an island instead of an inlet in one of two ways: it may make a full circuit of the island boundary and arrive back at a presumed inlet entrance point $E$ after moving along some number of capes and a fewer number of bays (Figure 19); or it may arrive back at the starting point $S_p$ before traveling along a bay that is not inside a D-inlet (Figure 20). In either case, it will have already covered all inlets created by the island boundary. It may be that not all of these inlets' doorways are locked, though, as shown in Figures 19 and 20. Therefore, before proceeding to cover the area around the island, the robot must lock these doorways. Each doorway will have been remembered as the grid line segment traveled along immediately prior to moving along a cape.

If the robot locks the doorway of an inlet containing the starting point $S_p$, it is prevented from returning to $S_p$ after covering the area $\mathcal{A}_1$ (Section 3.3). Therefore, it updates the location of $S_p$

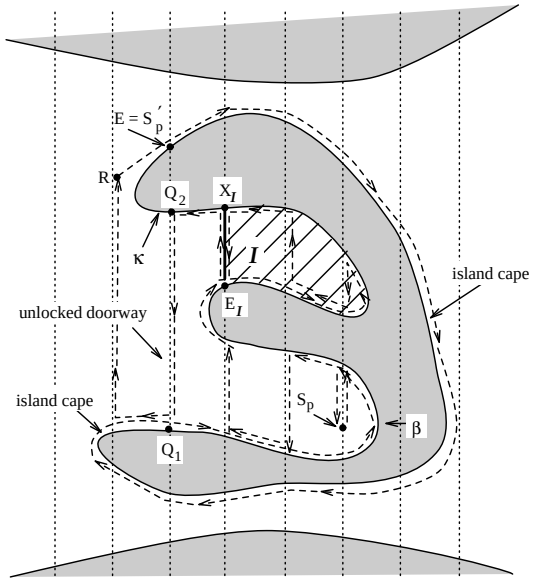



Figure 20: *Starting at point $S_p$, the robot zigzags along the grid lines until it encounters the entrance point $E_{\mathcal{I}}$ for inlet $\mathcal{I}$. It will then cover inlet $\mathcal{I}$ and, after locking its doorway $\overline{E_{\mathcal{I}}X_{\mathcal{I}}}$ (bold line segment) will continue to zigzag. At point $R$ it will notice the cape point $E$. It will then begin to follow the boundary of the island. When it arrives at the bay $\beta$, it will again begin to zigzag until it returns to $S_p$. At this point it knows that it has encountered an island and that $S_p$ is contained in one of the inlets created by the island's boundary. The grid line segment $\overline{Q_1Q_2}$ traveled along immediately prior to traveling along the cape $\kappa$ is considered an unlocked inlet doorway. By locking that doorway, the robot is prevented from returning to $S_p$. Therefore, $S_p$ is modified to be the point $E = S_p'$.*

to $S'_p$, where $S'_p$ is the inlet entrance point associated with the first cape the robot encountered for which there was no associated locked doorway when the robot returned to $S_p$ (Figure 20).

When the robot detects that it has encountered an island, it makes note of the endpoints of capes on the island boundary for which there are no associated locked doorways (and thus no associated bays). These are the island cape points.

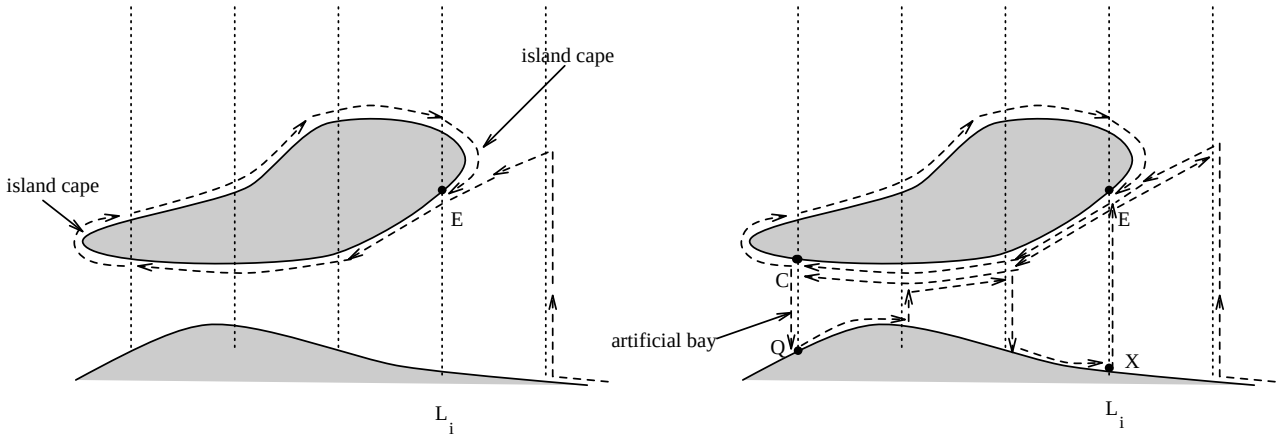
Using one of the island cape points as an *artificial bay point*, the robot creates an *artificial bay*. This bay is to be associated with the other island cape, thus creating an *artificial inlet*. An *artificial bay* is a section of a grid line lying in the interior of $\mathcal{A}_p$ and between two consecutive boundary intersection points, which the robot treats like a D-inlet bay. That is, after traveling along it, the robot may then begin to zigzag along the grid lines and cover the artificial D-inlet. The robot creates an artificial bay by noticing that it has revisited one of the island cape points, as explained below.

### 3.2.2 Covering the Area around an Island

By creating artificial D-inlets with artificial bays, the area surrounding an island may be covered in exactly the same way that D-inlets are covered. The only difficulty that arises is deciding which of the four island cape points is to be the artificial D-inlet's entrance point and which is to be the artificial bay point. Generally, the entrance point will be the first of the island's cape points encountered after detecting the island; the artificial bay point will be the first island cape point encountered after the entrance point.

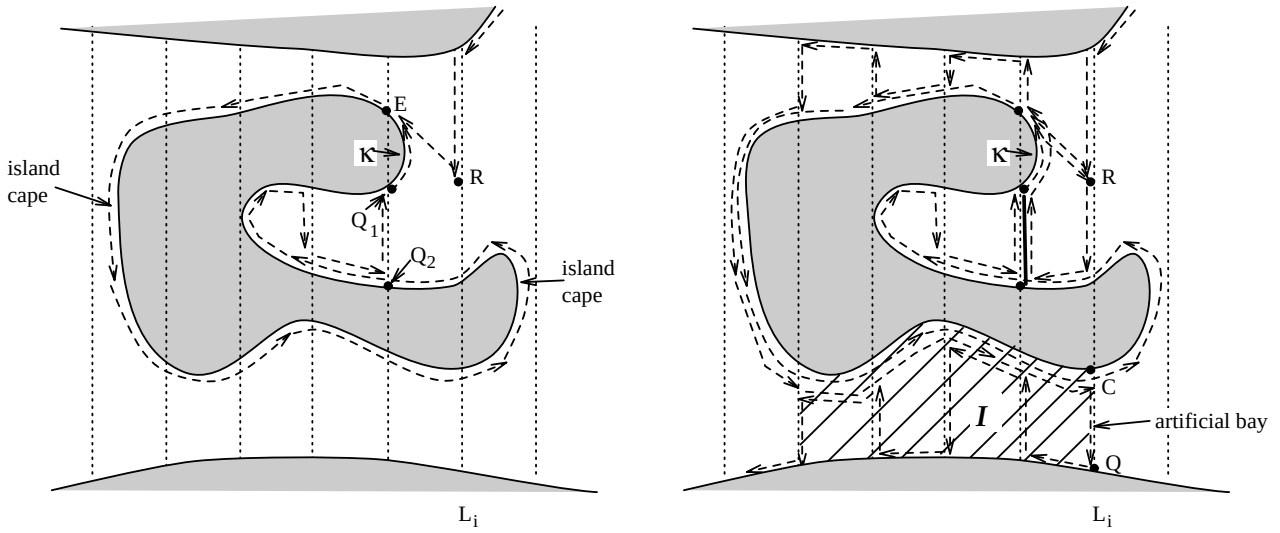
Assume the robot detects the island by making a full circuit of its boundary and returning to the cape point $E$. If $E$ is one of the island cape points, it is designated as the artificial inlet entrance point. After returning to $E$ and locking the doorways associated with the other capes on the boundary of the island, the robot continues to follow the boundary of the island as if it were entering a D-inlet with entrance point $E$. When it reaches one of the other island cape's endpoints, it recognizes that it is at an artificial bay point (Figure 21).

If $E$ is not one of the island cape points, then the robot must have detected the cape points in its sensing region. After returning to $E$, the robot will resume its motion as if it had not encountered the island. That is, the robot will simply move back to the point from which $E$ was detected and continue (Figure 22). When the robot encounters one of the island capes again after resuming its motion, the appropriate point is designated as the D-inlet entrance point (as for any other cape)




(a) (b)

Figure 21: *(a) The robot has arrived back at point $E$ and determined that it is on an island. (b) Since $E$ is an island cape point, the robot continues to follow the boundary of the island until it arrives at the island cape point $C$, the artificial bay point. The robot moves along the artificial bay $\overline{CQ}$ and then turns to zigzag back toward the artificial inlet's doorway $\overline{EX}$.*




(a) (b)

Figure 22: *(a) From grid line $L_i$, the robot senses the cape points $Q_1$ and $E$ of cape $\kappa$ and moves from its current position $R$ to the presumed D-inlet entrance point $E$. When it arrives back at $E$ after traveling around the island, it will lock the line segment $\overline{Q_1Q_2}$ traveled along immediately before returning to $E$. (b) The robot then moves back to $R$ and continues on as it was before. After it moves along the island cape on the left of the island, it will continue to follow the island boundary until it arrives at the next island cape point, $C$, which is the artificial bay point. The robot moves along the artificial bay $\overline{CQ}$ and then covers the artificial inlet $\mathcal{I}$ as it would any other D-inlet.*

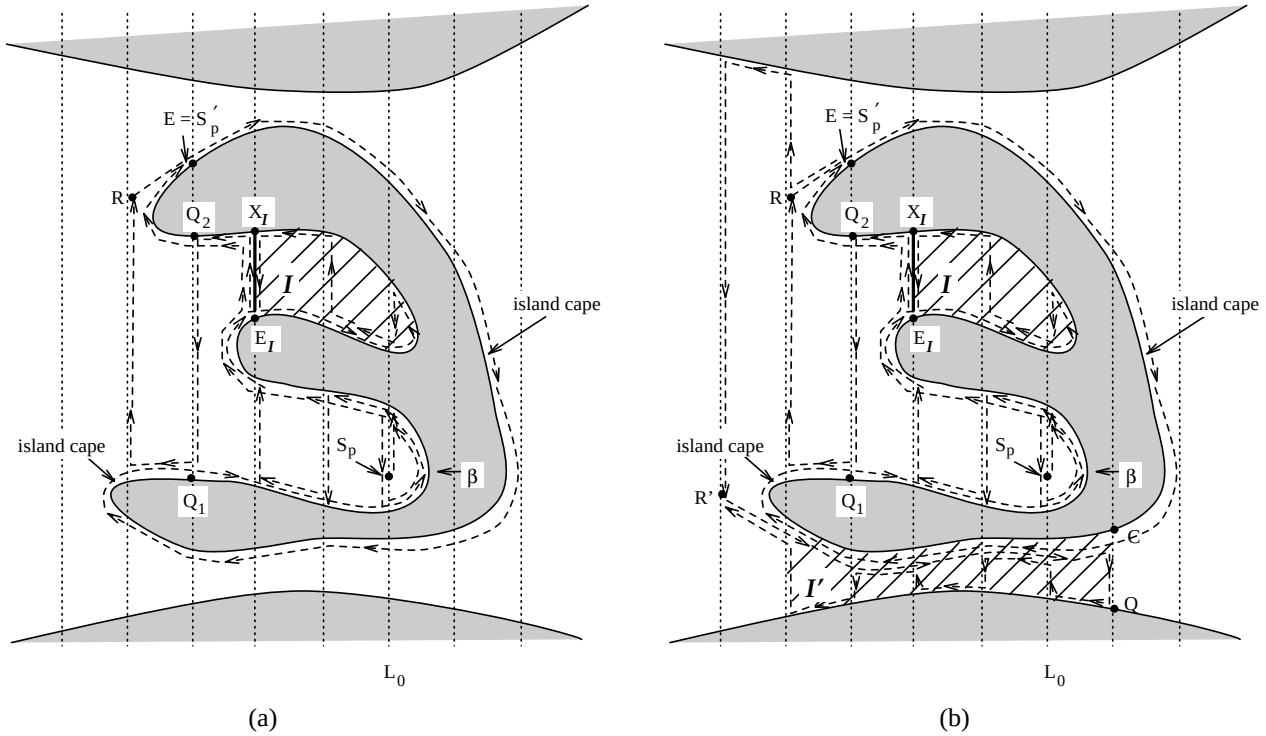



Figure 23: *(a) After the robot returns to $S_p$ and determines that it has encountered an island (Figure 20), it will continue to move along $L_0$ and then follow the boundary of the island (including the locked doorway $\overline{E_I X_I}$) back to the point $E = S'_p$. (b) After moving back to $E$, the robot moves back to $R$ and continues to zigzag. At the point $R'$, an island cape is detected. The robot follows the boundary of the island until it reaches the next island cape point $C$, the artificial bay point. The robot moves along the artificial bay, $\overline{CQ}$ and covers the artificial inlet $\mathcal{I}'$ as it would any other inlet.*

and the robot will begin to follow the boundary of the island. The first of the other island cape's endpoints the robot encounters while following the boundary will be designated as the artificial bay point.

If the robot detects that it has encountered an island by returning to the starting point $S_p$, it must first exit the locked inlet containing $S_p$. To do this, it will move along $L_0$ until it encounters the island boundary again and then follow the island boundary in the same direction it followed it before returning to $S_p$. The robot will follow the island boundary until it reaches the point $S'_p \in L_i$, which is the next cape point detected that is not part of a locked doorway. If this point is an island cape point, the robot will proceed as described above for the case when the robot arrived back at an island cape point after making a full circuit of the island's boundary. If the point $S'_p$ is not an

island cape point, the robot will proceed as if it had not encountered the island. That is, if it first moved to $S'_p$ along the boundary while zigzagging, it will continue to zigzag by moving along $L_i$ away from $S'_p$ as if it were still following the boundary. If $S'_p$ was first detected within the robot's sensing region, the robot will move back to the point from which $S'_p$ was originally detected and continue (Figure 23). When the robot encounters one of the island capes again, it designates the appropriate point as a D-inlet entrance point, moves along the boundary of the island until it encounters another island cape point, and designates this point as the artificial bay point.

Upon reaching an artificial bay point in any of these cases, the robot will then proceed along the artificial bay and cover the artificial inlet as it would any other D-inlet. Upon exiting and locking an artificial inlet, the artificial bay is converted into a locked doorway to prevent the robot from covering the area around the island a second time.

The procedure $CoverDInletOrIsland$ shown in Figure 24 is a modified version of the $CoverDInlet$ procedure (Figure 15); it incorporates the steps for detecting islands that cross grid lines in a non-simply connected area and creating artificial bays in order to cover the area around them.

### 3.3 Returning to the Starting Point

Whether in a simply connected or nonsimply connected area $\mathcal{A}_p$, when the robot first travels along a bay that is not inside a D-inlet, it will have covered the area $\mathcal{A}_1$. It must then return to $S_p$ (either the original starting point or as modified by procedure $CoverDInletOrIsland$, described in Section 3.2.2), in order to cover area $\mathcal{A}_2$ (Figure 9). This it does by simply continuing to follow the boundary after moving along the bay. Any locked doorways encountered while moving back to $S_p$ are treated as portions of the boundary: the robot moves from one doorway endpoint to the other and then away from the inlet boundary. If any new D-inlet entrance points are encountered while moving along the boundary, the robot covers these before proceeding. If the robot encounters the grid line $L_0$ before the point $S_p$ while moving along the boundary in this way, it moves along $L_0$ toward $S_p$ (Figure 25). This procedure for returning to the starting point is shown in Figure 26. After arriving at $S_p$, the robot begins to cover area $\mathcal{A}_2$ in the same fashion as $\mathcal{A}_1$.

**Input:**
$\mathcal{I}$: the inlet to be covered;
**Local Variable:**
$S'_p$ : the first presumed inlet entrance point encountered while moving around an island
for which the robot did not encounter a doorway before returning to $S_p$;

**procedure** *CoverDInletOrIsland*
{
Move to $E_{\mathcal{I}}$;
Store $E_{\mathcal{I}}$;
do
Move along boundary;
if a D-inlet entrance point is detected then
*CoverDInletOrIsland*();
until robot has moved along a bay or reached an artificial bay point
or returned to an inlet's entrance point or returned to $S_p$;
if robot moved along a bay then
*CoverArea*();
else if an artificial bay point is reached then
*CoverArea*();
Convert artificial bay to locked doorway;
else (an island is detected)
Lock doorways for capes other than the island capes;
if robot is at $S_p$ then
Move along $L_0$ until boundary is hit;
Move along boundary until $S'_p$ is encountered;
$S_p \leftarrow S'_p$;
if robot is at an island cape point then
do
Move along boundary;
until island cape point is reached;
*CoverArea*();
else
Remember island cape points as potential artificial bay points;
Return to point from which $E_{\mathcal{I}}$ was detected;
$\mathcal{I} \leftarrow$ NULL;
}

Figure 24: *The procedure for covering inlets and islands in a nonsimply connected area is shown. After encountering a cape on the boundary, the robot assumes it is entering a D-inlet until it discovers otherwise. This procedure incorporates the means of detecting an island and its artificial bay point. For a nonsimply connected area, calls to this procedure should replace calls to CoverDInlet in the procedure CoverArea.*

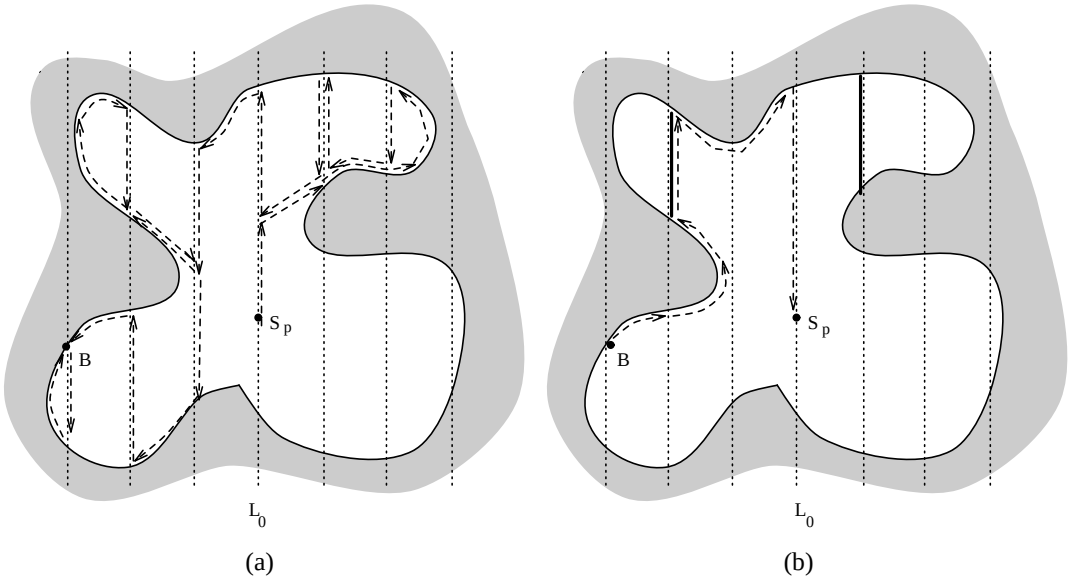



Figure 25: *(a) The robot covers area $\mathcal{A}_1$. Upon reaching bay point $B$ the second time, the robot realizes it must return to $S_p$. (b) The path the robot follows back to $S_p$. The locked doorways of D-inlets already covered are shown as thick line segments. When the robot encounters a locked doorway, it moves along the doorway as if it were a part of the boundary. When the robot reaches grid line $L_0$, it moves along it to the point $S_p$.*

```
procedure ReturnToS
{
    do
        Move along boundary;
        if locked doorway is encountered then
            Move along doorway;
            Turn away from locked inlet's boundary;
        if a D-inlet entrance point is detected
            CoverDInletOrIsland();
    until reach L0 or Sp;
    Move along L0 until Sp is reached;
}
```

Figure 26: *The procedure for returning to the starting point $S_p$ after area $\mathcal{A}_1$ has been covered.*

# 4 Motion in Three Dimensions

## 4.1 Motion within a Grid Plane

For an AUV to use the planar terrain-covering algorithm of Section 3 in the three-dimensional environment assumed in our model, it cannot simply move along grid lines in the $xy$-plane. It will instead move within the vertical grid planes that contain each grid line, following the slope of the terrain to maintain a certain distance from the surface. As stated in Section 2, the limits on the distance the robot maintains from the surface are determined in part by the parameters of its camera. We assume this simple method of three-dimensional terrain exploration since the two-dimensional algorithm extends easily in this case. If this assumption is not valid, a different approach may be necessary.

Since, from any position, the distance to different points of the ocean floor within the robot's sensed area is not the same, the robot must estimate its distance from the floor using an average of its sensor readings. The robot will adjust its height so as to maintain that average at about $h/2$ from the floor. The use of this average prevents the robot from being overly sensitive to small fluctuations in the terrain. Similarly, the robot must use an average slope to determine its instantaneous direction of motion.

How the average distance and slope are computed may depend on the particular application. For example, one way to compute them is by using all points in the imaged area that are within a certain range of vertical distances from the robot. If this range of distances is small, the robot will be sensitive to small decreases in the distance to the lowest point; if the range of distances is large, it will be less sensitive to small perturbations in the height of the floor but may have to compensate for this by moving closer to the floor.

Let $A_0$ denote the set of points that are used to compute the robot's average vertical distance. An example of the robot's motion when the range of values over which the average taken is small and includes the lowest point in its imaged area is shown in Figure 27. From position $R_0$, the robot moves along a horizontal line, maintaining distance $h/2$ from the flat surface. It continues to move horizontally until it reaches the point $R_1$, when it notices the surface begin to slope downward. Notice that it does not adjust its height in response to the small bump $B$ in the surface. The points on the bump are not in the area $A_0$, since, from every point at which this bump is sensed, there are other points in the sensed area with lower $z$ coordinates. Along the path segment $\overline{R_1R_2}$, the robot

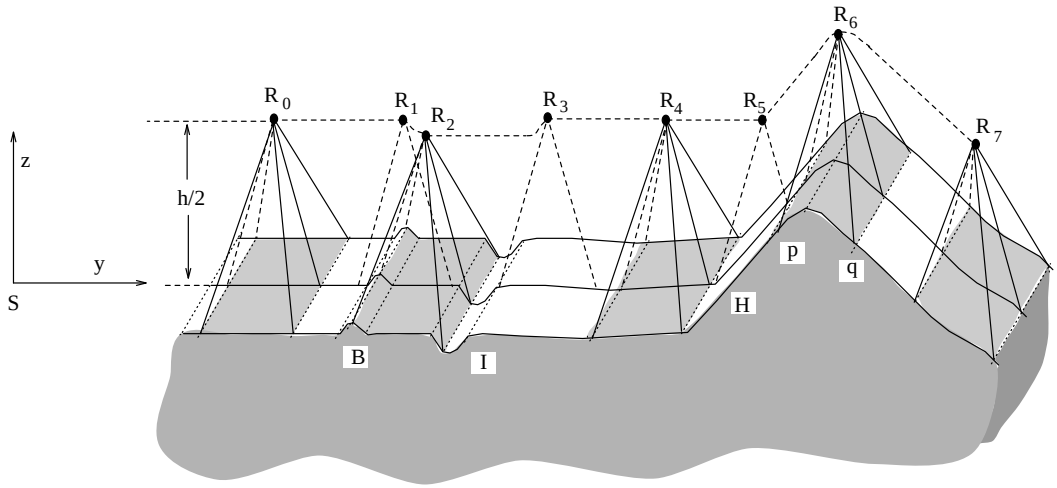



Figure 27: *The path the robot follows to maintain an average vertical distance of $h/2$ from the lowest points on the floor. The robot does not adjust its height in response to small rises in the surface ($B$) but responds immediately to any indentation ($I$). For more significant rises, or hills, in the surface ($H$), the robot adjust its height in response to this rise only when some point on the hill is in $A_0$. When the surface slopes in different directions at the points in $A_0$, such as at the point $R_6$, the slope of the path of the robot will be the average of these slopes.*

moves down following the slope of the indentation to maintain its vertical distance of $h/2$. It moves horizontally while the lowest points in the indentation are within its imaged area then moves up to the point $R_3$ to follow the slope on the opposite side of the indentation. Again, it moves horizontally at distance $h/2$ from the flat surface until the point $R_5$ at which it begins to follow the slope of the hill $H$. Note that the robot first detects the rise in the floor $H$ at point $R_4$, but it does not respond to this rise until the flat portion of the floor is no longer within its imaged area. When the robot is atop the hill at point $R_6$, the surface slopes in different directions at the points in $A_0$. In particular, the slope at the point $p$ is positive and at the point $q$ it is negative. The slope of the robot's path is the average of the slopes.

## 4.2 Motion Between the Grid Planes

To move from one grid plane to the next, the robot must follow the boundary $\mathcal{B}$. This is done in one of two ways, depending on whether the portion of the boundary is defined by a threshold surface or by the threshold slope (Section 2). If the robot reaches one of the threshold surfaces while moving in a certain grid plane $P_i$ (*i.e*, there is a point $p = (p_x, p_y, p_z)$ in its imaged area such that $p_z = z_{\min}$

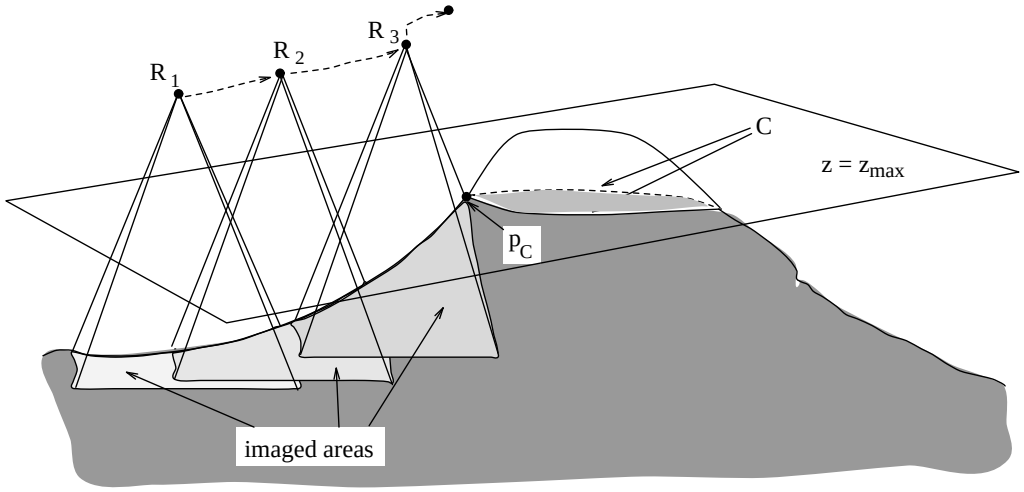



Figure 28: *The curve $C$ is one of the boundary curves for the area $A$. It represents the intersection of the ocean floor with the upper threshold surface $z = z_{\max}$. At the point $R_3$, the robot detects the point $p_C$ in its imaged area that is on this curve. It then turns to follow the boundary and moves parallel to the threshold surface, keeping the points on the curve $C$ in its imaged area.*

or $p_z = z_{\max}$), it will follow the boundary by moving parallel to the threshold surface away from $P_i$ toward $P_{i+1}$ or $P_{i-1}$ as the algorithm dictates (Figure 28).

On the other hand, the robot may determine that it has reached the boundary because the rate of change in its height $z$ in the direction of maximal increase is equal to $\mu$. In this case, it will follow the boundary by using its sensors to determine the contour of the floor along which the threshold slope is reached. It follows this contour in the direction leading to the next grid plane (Figure 29). Again, the robot must use averaged sensor readings to determine the general slope of the floor in its sensing region.

In summary, the motion planning algorithm for a three-dimensional, projectively planar environment differs little from that for a planar environment. The projection of the path the robot follows in the three-dimensional environment $\mathcal{A}$ is the path it would follow in the corresponding planar environment $\mathcal{A}_p$. In $\mathcal{A}_p$, the robot zigzags from one grid line to the next; in $\mathcal{A}$, this same $x$ and $y$ motion leads the robot from one grid plane to the next. The difference is that the $xy$-motion is augmented by vertical movement within each grid plane to account for changes in the height of the terrain.

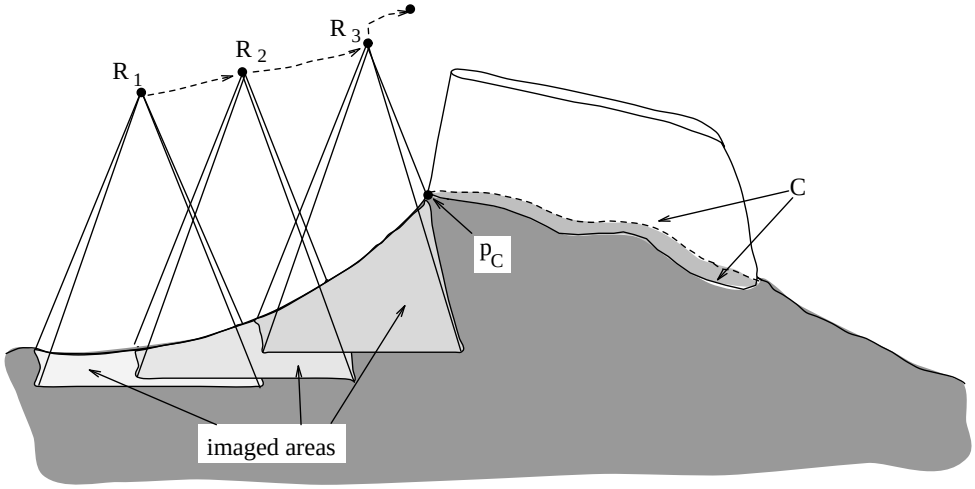



Figure 29: *The curve $C$ represents the contour along the floor at which the slope of the surface is equal to $\mu$. This is one of the boundary curves for the area $\mathcal{A}$. At the point $R_3$, the robot detects a point $p_C \in C$ and turns to follow this boundary by maintaining the points on the curve $C$ in its imaged area.*

# 5 Algorithm Complexity

The complexity of the algorithm presented is measured in two different ways: in terms of the distance traveled by the robot and in terms of the amount of memory required to store the input information. We show below that the length of the robot's path is, in the worst case, linear in the lengths of the outer boundary and the island boundaries and in the length of the grid line segments in the interior of $\mathcal{A}_p$. The upper bound on the robot's path length is first developed for the planar environment; the upper bound in a nonplanar environment follows as a corollary to this result. It is also shown that, if the boundary $\mathcal{B}_p$ is described by a semi-algebraic set, the amount of memory required by the robot in a planar or nonplanar environment is linear in the size of this description.

In what follows, let $P$ be the total path length of the robot in a planar environment, $P'$ the length of the planar outer boundary curve, and $P''$ the sum of the lengths of the planar island boundaries. Also, let $L'$ be the sum of the lengths of all the grid line segments in the interior of $\mathcal{A}_p$ that are not inlet doorways, and let $L''$ be the sum of the lengths of all inlet doorways. Further, let $Q'$ be the sum of the lengths of all capes on the boundary $\mathcal{B}_p$.

**Theorem 5.1** *The path length $P$ of the robot following the planar terrain-covering algorithm satisfies*

*the inequality*

$$P \leq L' + 3L'' + 2P' + 3P'' + 2Q'. \tag{1}$$

*Proof:* The robot's path is generated by the four procedures $CoverArea$, $CoverDInletOrIsland$, $ExitDInlet$, and $ReturnToS$ described in Section 3. For each of these procedures, we provide an upper bound on the length of the path generated. The theorem follows as the sum of these path length estimates.

The procedure $CoverArea$ causes the robot to zigzag along the grid lines. In this process, each grid line in a given area is traveled exactly once and each boundary section is traveled no more than once. (Some portions of the boundary will not be traveled along at all.) This is assured because if the robot revisits a given grid line, this means it must have traveled along a section of boundary and arrived back at the given grid line without crossing another grid line. That is, it must have traveled along a bay. When the robot travels along a bay, the procedure $CoverArea$ stops. Every portion of a given area (with the exception of D-inlet doorways) is covered by a call to the $CoverArea$ procedure. Since D-inlet doorways are locked after the inlets have been covered, no area is covered more than once by this procedure. Thus the contribution to the robot's path length from this procedure is no more than $L' + L'' + P' + P''$.

For each call to the procedure $CoverDInletOrIsland$, the robot must first move to the D-inlet entrance point if it is not already there. The distance traveled in doing this will be no more than the length of the cape that causes the inlet. For all inlets, this amounts to no more than $Q'$ in path length. After moving to the entrance point, the robot then either moves along the boundary of the D-inlet until it reaches the inlet bay or moves around the boundary of the island until it returns to the entrance point. This contributes no more than $P' + P''$ to the robot's path length. Once each island has been detected, part of its boundary will be traveled along once again as the boundary of an artificial inlet. This requires the robot to travel a distance of no more than $P''$. Note that if the robot's starting point is contained in an island inlet and it must follow the boundary of the inlet to the new starting point $S'_p$, this portion of the island boundary will not be part of the boundary of the island's artificial inlet. This contribution to the robot's path length is therefore included in the estimate $P''$. Since each inlet covered by this procedure is locked after it has been covered, this procedure will be called only once for each D-inlet, artificial or otherwise (and thus once for each cape). The total contribution to the robot's path length from procedure $CoverDInletOrIsland$ is

no more than $P' + 2P'' + Q'$.

When the robot exits each inlet using procedure $ExitDInlet$, it must move along the D-inlet doorway at least once. Note, however, that since $L''$ is the total length of all inlet doorways and since the procedure $CoverArea$ does not cause the robot to travel along any D-inlet doorways, the contribution to the path length for this single traversal of the doorway is already included in the estimate given for $CoverArea$. In some cases, $ExitDInlet$ causes the robot to travel the inlet doorway a second time, which, in total, adds no more than $L''$ to the robot's path length.

After covering the doorway in procedure $ExitDInlet$, if the robot did not travel along a cape to reach this inlet, it must move back to the point from which it detected the inlet entrance point. This distance will be no more than the length of the cape. Since the robot can not enter a D-inlet more than once, it also can not exit it more than once. Thus the total addition to the robot's path length from the $ExitDInlet$ procedure is no more than $L'' + Q'$.

Finally, the procedure $ReturnToS$ contributes no more than an additional $L''$ to the path length. This represents the maximum possible distance traveled along the locked D-inlet doorways that the robot encounters. In returning to $S_p$, the robot also travels along portions of the outer boundary and the island boundaries. However, this contribution to the path length is already included in the estimate given for the procedure $CoverDInletOrIsland$. To see why, note that when covering an inlet, the robot moves along the inlet boundary, but when returning to $S_p$ the robot never moves along an inlet boundary. Thus the sum of all these boundary segment traversals can be no more than the $P' + P''$ already included in the estimate given for $CoverDInletOrIsland$. Note also that the distance traveled along grid line $L_0$ to $S_p$ may also be considered included in the estimate $P'$ from $CoverDInletOrIsland$. To return to $L_0$ from some point in the area $\mathcal{A}_1$, the robot can not make a full circuit of the boundary (as is indicated by the estimate $P'$). The segment of $L_0$ it travels along to reach $S_p$ must be shorter than the section of the outer boundary it did not travel along.

Having shown path length estimates for all portions of the robot's path, the bound given in the theorem follows as the sum of these estimates. ■

The upper bound on the path length given in Theorem 5.1 requires at most 3 traversals of any boundary section in a nonsimply connected area. When the area $\mathcal{A}$ is simply connected, this upper bound is reduced to $L' + 3L'' + 2P' + 2Q'$. Further, if the area is simply connected and contains no inlets, the robot will travel along each grid line exactly once and the outer boundary no more than

twice. That is, the length of the robot's path is no more than $L' + 2P'$ in this case.

To determine the relative merit of our algorithm compared to other planar terrain-covering algorithms, the only candidate for comparison is the Seed Spreader algorithm of [Lumelsky, 1990], since, as mentioned in the introduction, it is the only other algorithm that handles arbitrarily shaped boundaries. The upper bound on the path length produced by the Seed Spreader algorithm is given as:

$$P \leq 2(L' + L'') + 2nw + 4(P' + P'') + 2nn_0p_{max} + p_{max}\Sigma_{j=1}^{r} n_j(n_j + 1). \quad (2)$$

where $n$ is the number of obstacles (which is proportional to the number of islands and inlets in our model) in the environment, $w$ is the horizontal spacing between grid lines, $n_o$ is the number of obstacles that cross a grid line, $n_j$ is the number of obstacles completely between grid lines $L_j$ and $L_{j+1}$, $r$ is the number of grid lines in the area $\mathcal{A}_p$, and $p_{max}$ is the maximum perimeter of an obstacle. The last term of inequality (2) accounts for the path length required to cover obstacles that lie completely between two grid lines but cannot be completely seen from one of these. Since we assume that no such obstacles exist in our environment, this term can be ignored in our comparison. The term $2nw$ of (2) accounts for the path length needed to divert to areas not covered while zigzagging and is roughly equal to the $2Q'$ in (1). These terms can therefore also be ignored.

Without knowing what percentage of the grid lines are doorways in the worst case, it is difficult to determine by a comparison of the relevant terms in (1) and (2) ($L'+3L''$ and $2(L'+L'')$, respectively) which algorithm results in shorter path length along the grid line segments in the environment. However, inequality (1) is the clear winner in the comparison of the number of times the polygonal boundary curves are traveled along ($2P'+3P''$ versus $4(P'+P'')$). Additionally, the term $2nn_op_{max}$ of inequality (2) has no counterpart in (1). It is present in (2) since, with the Seed Spreader algorithm each obstacle may, in the worst case, be traveled around once for every other obstacle in the environment. This can never happen in our case since we remember certain points on the boundary of each obstacle (inlet or island in our model) to prevent re-exploration of the obstacle. By remembering these few specific points, we are able to reduce significantly the path length of the robot. When measured in terms of the number of polygonal boundary curves, our algorithm is linear while the Seed Spreader algorithm is quadratic.

In a nonplanar, three-dimensional environment, the upper bound on the path length differs from the bound in the planar environment by no more than a multiplicative constant that is dependent

on the threshold slope, as shown by the following corollary to Theorem 5.1.

**Corollary 5.1** *The path length $\hat{P}$ of the robot following the terrain-covering algorithm in a three-dimensional environment satisfies the inequality*

$$\hat{P} \leq \sqrt{1+\mu^2}P,$$

*where $P$ is the length of the path in the corresponding planar environment.*

*Proof:* The maximum slope of the terrain over which the robot will travel is $\mu$. Therefore, no segment of the projected path can have a slope of more than $\mu$. For any vector $\hat{U} = (u_x, u_y, u_z)$ in the three-dimensional environment that projects onto the planar vector $U = (u_x, u_y)$, the following relation holds ;

$$u_z \leq \mu|U|,$$

therefore,

$$|\hat{U}|^2 \leq (1+\mu^2)|U|^2$$

from which the relationship in the corollary follows. ■

The amount of memory required by the robot is given as the maximum number of points for which the robot must remember coordinates. This we estimate in terms of the size of the boundary description. The bound shown applies whether the environment is planar or nonplanar. Let $\mathcal{B}_j, j = 1, \ldots, m$, represent the $m$ simple curves that constitute the boundary of $\mathcal{A}_p$. Assume the curves $\mathcal{B}_j$ can be described as semi-algebraic sets. In other words, each $\mathcal{B}_j$ can be obtained as a result of set-theoretic operations (intersections, unions, and complements) on a finite set of algebraic curves [Canny, 1988, Latombe, 1991]. This boundary description is used for the sake of argument only. It is never necessary for the robot to compute these curves.

Let $\lambda$ be the number of grid lines in the interior of $\mathcal{A}_p$ and $n$ the maximum number of connected boundary sections lying between any two grid lines. Assume that each such boundary section can be described by no more than $p$ algebraic curves of maximum degree $d$.

**Theorem 5.2** *The maximum number of points stored in the robot's memory is bounded above by $\lambda npd$.*

*Proof:* The only points the robot must remember for this algorithm are intersections between the boundary and the grid lines. Since a straight line can intersect a curve of degree $d$ in at most $d$ points, a given grid line cannot intersect the boundary in more than $npd$ points. Thus the total number of intersections of grid lines with the boundary can be no more than $\lambda npd$. ■

As an example, consider Figure 3. Assume that the three boundary curves shown there can each be described by 10 parabolas or circular arcs (a fairly complex description of the environment). In this case $p = 3 \times 10 = 30$ and $d = 2$. Further, assume that $\lambda = 20$ and $n = 5$. The maximum number of points the robot must store in its memory in this case is $\lambda npd = 20 \times 5 \times 30 \times 2 = 6000$.

# 6 Algorithm Correctness

The correctness of the algorithm presented in Sections 3 and 4 is established in this section. Due to lack of space, only the relevant statements of the lemmas, corollaries, and theorems are stated here. The text of the proofs may be found in [Hert, 1995].

In what follows, assume, without loss of generality, that a coordinate system has been chosen in the area $\mathcal{A}_p$ to be covered. Let $\mathcal{B}_j, j = 1, \ldots, m$, represent the $m$ simple closed curves that constitute the boundary of $\mathcal{A}_p$. For any given boundary curve $\mathcal{B}_j$ of area $\mathcal{A}_p$, let $\kappa(\mathcal{B}_j)$ be the number of capes on the curve and $\beta(\mathcal{B}_j)$ be the number of bays. Further, for a given portion $\mathcal{A}'$ of the area $\mathcal{A}_p$, let $\mathcal{I}(\mathcal{A}')$ be the number of D-inlets in the area $\mathcal{A}'$ and $\mathcal{B}(\mathcal{A}')$ be the boundary of $\mathcal{A}'$.

To prove that the algorithm is correct for a simply connected area, it is sufficient to show that any area without inlets is covered by the algorithm and that, for all other areas, every inlet will be covered. To establish the truth of these statements, the following relationships between the number of bays, capes, and D-inlets are shown.

**Lemma 6.1** *For any closed planar curve $\mathcal{B}_j$,*

$$\beta(\mathcal{B}_j) = \kappa(\mathcal{B}_j) + 2.$$

**Corollary 6.1** *For any open planar curve $\mathcal{B}' = (b_1 b_2)$ with $b_1, b_2 \in L_i$ for some $i$, such that the open grid line segment $\overline{b_1 b_2}$ does not intersect $\mathcal{B}'$,*

$$\beta(\mathcal{B}') = \kappa(\mathcal{B}') + 1.$$

Corollary 6.1 implies that the boundary of each of the areas $\mathcal{A}_1$ and $\mathcal{A}_2$ and of each inlet contains one more bay than cape. The following lemma establishes that the number of D-inlets in a given area is equal to the number of capes on the boundary of the area.

**Lemma 6.2** *For any simply connected area $\mathcal{A}'$,*

$$\mathcal{I}(\mathcal{A}') = \kappa(\mathcal{B}(\mathcal{A}')).$$

Given the relationships shown in Corollary 6.1 and Lemma 6.2, the following lemma shows that if $\mathcal{A}_1$ and $\mathcal{A}_2$ contain no inlets, they will each be covered by the algorithm. It also establishes that any D-inlet that contains no subinlets will be covered.

**Lemma 6.3** *Let $\mathcal{B}' = (b_1 b_2)$ be an open planar curve such that $b_1, b_2 \in L_i$ for some $i$, and the open grid line segment $\overline{b_1 b_2}$ does not intersect $\mathcal{B}'$. If the curve $\mathcal{B}'$ contains only one bay, the area bounded by $\mathcal{B}'$ and $\overline{b_1 b_2}$ will be covered by a robot beginning at either $b_1$ or $b_2$ and following the algorithm.*

Since we assume that every straight line intersects the boundary in only a finite number of points, the following finiteness condition holds.

**Lemma 6.4** *For any area $\mathcal{A}'$, the number of capes on its boundary, $\kappa(\mathcal{B}(\mathcal{A}'))$, is finite*

Lemmas 6.2 and 6.4 imply that there are a finite number of inlets in any given area. The following lemma shows that each will be covered.

**Lemma 6.5** *In a given simply connected area $\mathcal{A}''$, every D-inlet will be covered by a robot following the algorithm.*

Lemmas 6.3, 6.4, and 6.5 provide the basis for the inductive argument that proves the following theorem.

**Theorem 6.1** *Any simply connected area $\mathcal{A}_p$ will be covered by a robot following the algorithm.*

For a nonsimply connected area that contains islands that cross at least one grid line, we show the following relationships between the number of bays and capes on each island's boundary as a second corollary to Lemma 6.1.

**Corollary 6.2** *For every curve $\mathcal{B}_j$ that is the boundary of an island,*

$$\beta(\mathcal{B}_j) = \kappa(\mathcal{B}_j) - 2.$$

This establishes the presence of the island capes that give rise to artificial inlets. By creating an artificial inlet for a given island, the robot essentially joins the island to another portion of the boundary (through the artificial bay), thus effectively eliminating the island. By this means a nonsimply connected area containing $k$ islands is converted into an area containing $k-1$ islands. Since Theorem 6.1 establishes that every area containing 0 islands will be covered, this provides the basis for an inductive argument on the number of islands in a nonsimply connected area. This argument is then used to prove the following theorem.

**Theorem 6.2** *Any nonsimply connected area $\mathcal{A}_p$ will be covered by a robot following the algorithm.*

This establishes the correctness of the algorithm presented in Sections 3 and 4 for simply and nonsimply connected areas.

# 7 Example and Discussion

In this example, the algorithm's performance is demonstrated using the nonsimply connected area shown in Figure 30. Planes $P_{-4}, \ldots, P_4$ are the grid planes that intersect the surface to be covered. The planes $z_{\max}$ and $z_{\min}$ are the threshold surfaces. The boundary of the area in this example is defined by the intersection of the floor with the plane $z_{\max}$. The robot's starting point $S$ is chosen arbitrarily.

The path of the robot is shown in panels (a) through (j) (Figures 31 and 32) projected on the the $xy$-plane. The robot's path is shown with dashed lines. Locked doorways are shown as thick line segments.

While moving along $L_0$, the robot notices a D-inlet entrance point and moves to it (Figure 31, panel (a)). It begins to follow the boundary of the inlet and in doing so, enters another D-inlet. It follows the boundary of this inlet until it reaches the inlet bay. It covers the inlet, locks its doorway and the continues to follow the boundary (Figure 31, panel (b)).

When it reaches the next bay, the robot zigzags along the grid lines until it arrives at an inlet

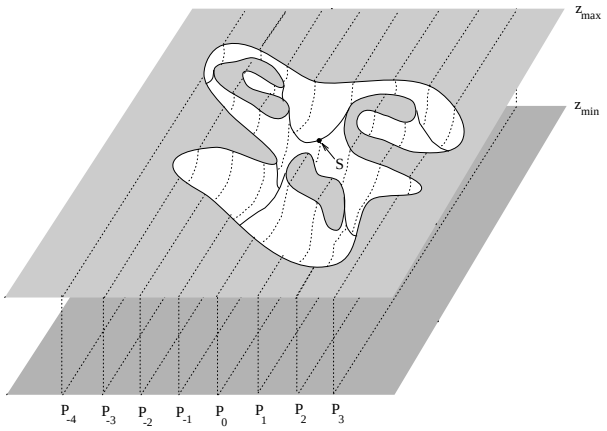



Figure 30: A sample nonplanar environment for a robot to cover.

doorway. It locks this doorway and moves back to the point from which it detected the entrance point (Figure 31, panels (c1) and (c2)).

While zigzagging, the robot notices cape points on an island and moves to one of the cape points as if entering a D-inlet. It follows the boundary of the island until it travels along the bay on the island's boundary. After traveling along the bay it is at the D-inlet doorway, so it locks the doorway and moves back to the point from which it detected the entrance point and continues (Figure 31, panel (d)). From grid line $L_{-4}$, the robot detects another presumed D-inlet entrance point. It moves to this cape point and follows the boundary of the island until it returns to the cape point (Figure 31, panel (e)).

Knowing that it is on an island and at an island cape point, the robot continues to follow the boundary of the island until it reaches another island cape point (*i.e*, the artificial bay point). It then moves along the artificial bay and zigzags back to the doorway of the artificial inlet. Before returning to grid line $L_{-4}$, it locks the artificial inlet's doorway and bay (Figure 32, panel (f)). When the robot travels along the bay with endpoints on $L_{-4}$, it determines that it must return to $S_p$. To return to $S_p$, the robot continues to follow the boundary after moving along the bay. When it moves along a cape to a D-inlet entrance point, it enters the D-inlet by continuing to follow the boundary. Upon reaching the bay on the D-inlet's boundary, it begins to zigzag back toward the doorway. The doorway is locked after it has been covered (Figure 32, panel (g)).

After locking the doorway, a cape on the second island is detected. The robot moves to the lower

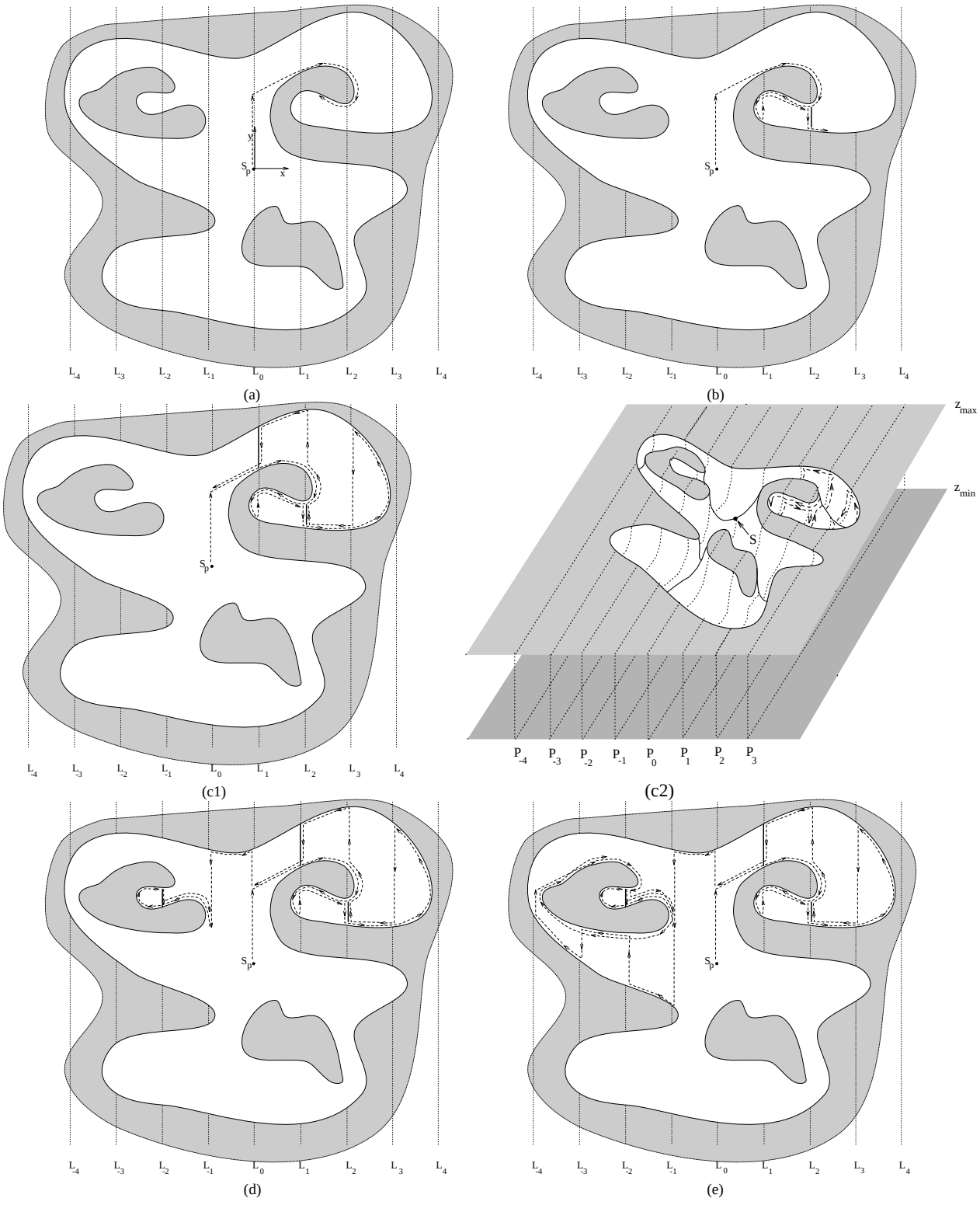



Figure 31: The first five snapshots of the robot's path in the environment of Figure 30.

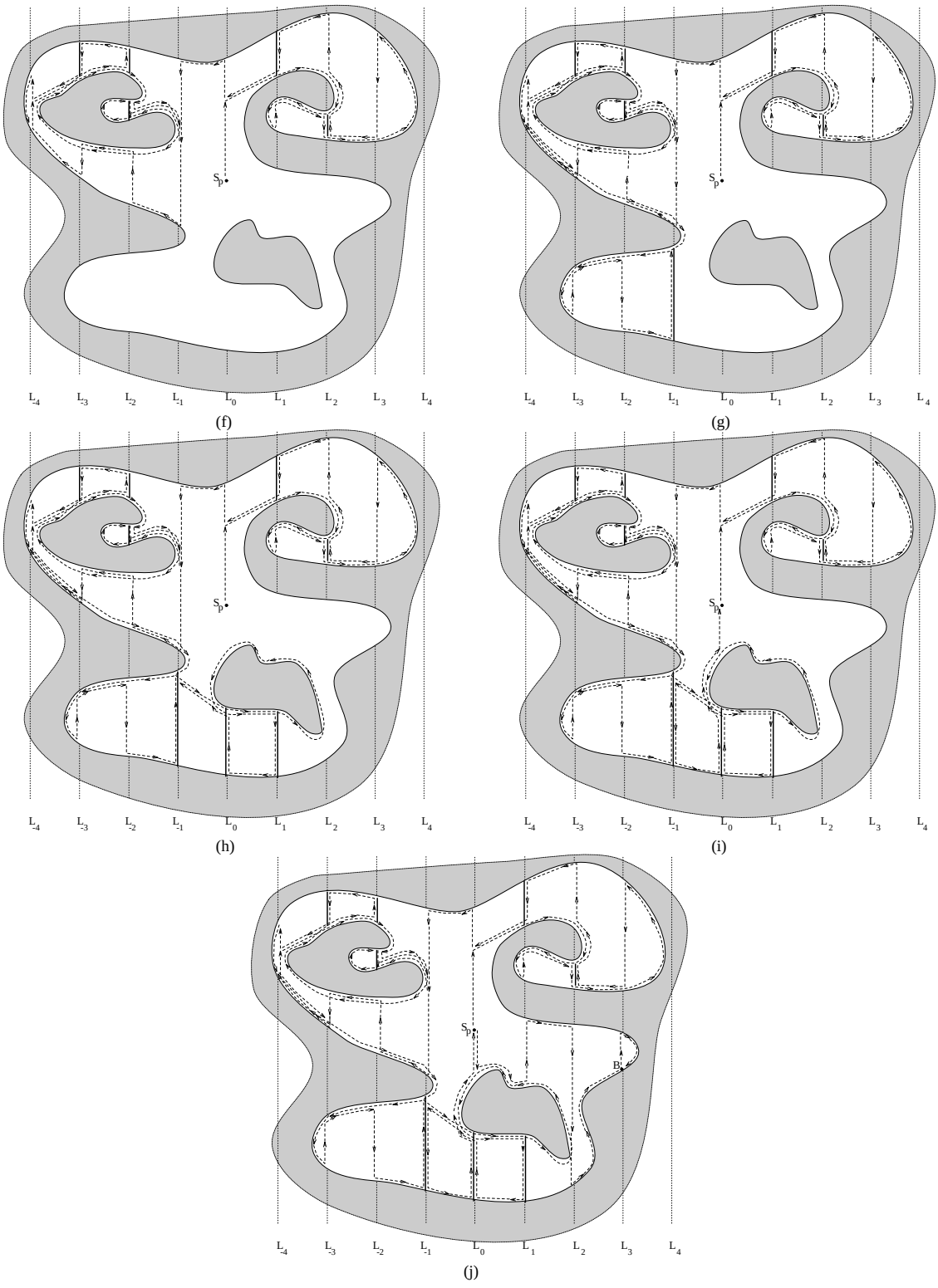



Figure 32: The last five snapshots of the robot's path in the environment of Figure 30.

cape point, which it assumes to be a D-inlet entrance point and then moves around the island's boundary. After making a full circuit of the island, the robot continues to follow the island boundary in order to enter the island's artificial inlet. It reaches the artificial bay point, zigzags back to the doorway, exits the artificial inlet, and locks the doorway and bay (Figure 32, panel (h)). Then the robot continues to follow the boundary in order to return to $S_p$. It moves along the locked doorways of the inlets just covered and around one of the island capes. When it reaches $L_0$ again, it moves along it until it reaches the point $S_p$ (Figure 32, panel (i)).

Upon reaching $S_p$, the robot moves in the direction opposite to its original direction at $S_p$ and zigzags to cover the rest of the area, finishing at bay point $B$ (Figure 32, panel (j)).

If $P_o$ represents the sum of all grid line segments in the interior of the area $\mathcal{A}_p$ and half the length of all boundary curves, this is an approximation of a lower bound on the robot's path length, for the chosen coordinate system. It represents, roughly, the path length necessary to zigzag along the grid lines in $\mathcal{A}_p$. In this example, since the area contains inlets and islands, $P_o$ is not achievable and thus does not represent the greatest lower bound. The path generated by our algorithm, with no knowledge of the environment, is approximately $1.7P_o$. This could be reduced to approximately $1.5P_o$ if the robot were to start at a bay point and knew it was doing so. In this case, it would not be necessary for it to return to $S$, since $\mathcal{A}_1 = \mathcal{A}_p$. The incorporation of any further information about the environment will lead to little improvement in the path length of the robot, as long as that information is only partial. If the robot were equipped with information about the location of islands, capes, and bays, the procedures for detecting these phenomena would be trivial. However, even with this additional information, the robot would have to enter inlets and travel around the boundaries of islands in order to cover them. The only possible area for improvement would be in the order in which the inlets and islands are visited. To avoid backtracking, these should be covered in the order in which they are encountered, which is exactly what our algorithm dictates. Thus, if information about the locations of these boundary sections were available, it could not be used to improve significantly the path length of the robot.

It is possible that, with a different coordinate system (which would lead to a different orientation of the grid lines) the path length could be reduced. However, finding the optimal coordinate system would not only require complete knowledge of the environment but would also require an exhaustive search of the possibilities.

Perhaps the most restrictive assumption in our model is that the robot have a single, downward-pointing camera. By relaxing this assumption, the algorithm might be used to cover more general terrains (*e.g.*, terrains that are not necessarily projectively planar or that contain more steeply sloping hills). However, to guarantee coverage of the entire terrain, additional assumptions about the position, range, and motion of the cameras must be made.

## References


[Blidberg, 1995] Blidberg, D. R. and Jalbert, J., 1995, "Mission and System Sensors", In *Underwater Robotic Vehicles: Design and Control, Edited by J. Yuh*, TSI Press, Albuquerque, 1995, pp. 185 – 220.

[Burke, 1992] Burke, S. E. and Rosenstrach, P. A., 1992 "High-resolution monopulse piezopolymer sonar sensor", In Proceedings 1992 IEEE Symposium on Autonomous Underwater Vehicle Technology, pp. 209 – 214.

[Canny, 1988] Canny, J. H., 1988, *The Complexity of Robot Motion Planning*, MIT Press, Cambridge, MA, 1988.

[Choi, 1995] Choi, S. K., Yuh, J., and Takashige, G. Y., 1995, "Design of an Omni-Directional Intelligent Navigator", In *Underwater Robotic Vehicles: Design and Control, Edited by J. Yuh*, TSI Press, Albuquerque, 1995, pp. 277 – 298.

[Chu, 1992] Chu, J. S., Lieberman, L. A., and Downes, P., 1992, "Automatic camera control for AUVs: a comparison of image assessment methods", In Proceedings 1992 IEEE Symposium on Autonomous Underwater Vehicle Technology, pp. 191 – 201.

[Gordon, 1992] Gordon, A., 1992, "Use of laser scanning systems on mobile underwater platforms", In Proceedings 1992 IEEE Symposium on Autonomous Underwater Vehicle Technology, pp. 202 – 205.

[Haywood, 1986] Haywood, R., 1986, "Acquisition of a micro scale photographic survey using an autonomous submersible", In Proceedings 1986 IEEE Oceans, Vol 5, pages 1423 – 1426.

[Henriksen, 1994] Henriksen, L., 1994, "Real-time Underwater Object Detection Based on Electrically Scanned High-resolution Sonar", In Proceedings 1994 IEEE Symposium on Autonomous Underwater Vehicle Technology, pp. 99 – 104.

[Hert, 1995] Hert, S., Tiwari, S., and Lumelsky V., 1995, A Terrain-Covering Algorithm for an AUV, University of Wisconsin–Madison Robotics Laboratory, Technical Report RL-95001.

[Latombe, 1991] Latombe, J.-C., 1991, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, pp. 146 – 149.

[Lozano-Pérez, 1983] Lozano-Pérez, T., 1983 "Spatial Planning: A Configuration Space Approach", *IEEE Transactions on Computers*, C-32(2): 108 – 120.

[Lumelsky, 1990] Lumelsky, V. J., Mukhopadhyay, S., and Sun, K., 1990, "Dynamic Path Planning in Sensor-Based Terrain Acquisition", *IEEE Transactions on Robotics and Automation* 6(4): 462 – 472.

[Marks, 1994] Marks, R. L., Rock, S. M., and Lee, M. J., 1995, "Real-Time Video Mosaicking of the Ocean Floor". *IEEE Journal of Oceanic Engineering* 20(3): 229 – 241.

[Oommen, 1987] Oommen, B. J., Iyengar, S. S., Rao, N. S. V., and Kashyap, R. L. 1987, "Robot Navigation in Unknown Terrains Using Learned Visibility Graphs. Part I: The Disjoint Convex Obstacle Case", *IEEE Journal of Robotics and Automation* RA-3(6): 672 – 681.

[Rao, 1990] Rao, N. S. V. and Iyengar, S. S., 1990 "Autonomous Robot Navigation in Unknown Terrains: Incidental Learning and Environmental Exploration", *IEEE Transactions on System, Man and Cybernetics* 20(6): 1443 – 1449.

[Rosenblum, 1992] Rosenblum, L. and Kamgar-Parsi, B., 1992, "3-d reconstruction of small underwater objects using high-resolution sonar data", In Proceedings 1992 IEEE Symposium on Autonomous Underwater Vehicle Technology, pp. 228 – 235.