Range-Sensor Based Navigation in Three Dimensions

Ishay Kamon
Dept. of Computer Science
Technion

Elon Rimon
Dept. of Mechanical Engineering
Technion

Ehud Rivlin
Dept. of Computer Science
Technion

Abstract

We present new results in range-sensor based navigation of a point robot in three-dimensional polyhedral environments, and incorporate them into a new globally convergent navigation algorithm. We show that the entire surface of a polyhedral obstacle is visible from a finite number of view points, which are located on convex edges within the convex hull of the obstacle. We introduce the notion of a locally shortest path in three dimensions, investigate its properties and present a novel method for estimating this path. Based on these results we describe a novel data structure termed the Convex Edges Graph or CEG, which consists of convex obstacle edges and supports both surface exploration and calculation of shortest paths. We then describe the new sensor-based navigation algorithm in three-dimensions, termed 3DBug.

The 3DBug algorithm strives to process the sensory data in the most reactive way possible, without sacrificing the global convergence guarantee. 3DBug is the first Bug-type algorithm which uses three-dimensional range data and plans three-dimensional motion throughout the navigation process. During motion towards the target the robot follows the locally shortest path in a purely reactive fashion. During traversal of an obstacle surface, which is the second motion mode of the algorithm, the robot incrementally constructs the CEG of the obstacle being followed, while performing local shortcuts based on range data. Simulations show that 3DBug generates paths which resemble the globally shortest paths in simple scenarios. Moreover, the algorithm generates reasonably short paths even in concave, room-like environments.
1 Introduction

Autonomous robots navigating in a realistic setting must use sensors to perceive the environment and plan accordingly. Sensor-based motion planning approaches use either global or local planning, or a combination of both. In the global approach, the robot builds a global world model based on sensory information (see for example [5, 7, 23, 25]). Global path-planners guarantee that either the target is reached or the robot concludes that the goal is unreachable. However, the tasks of constructing a sensor-based global model and localizing the robot relative to the model impose a heavy computational burden on the robot, due to the inherent uncertainty in the sensor data. In contrast, local path-planners typically work in a purely reactive fashion, using navigation vector-fields which directly map the sensor readings to actions (see for example [10, 22]). However, the local planners do not guarantee global convergence to the target.

We focus on a midway approach, called the Bug approach, originated by Lumelsky and Stepanov [17], and subsequently studied by [16, 18, 24]. The Bug approach minimizes the computational burden on the robot while still guaranteeing global convergence to the target. The basic Bug algorithms consist of two reactive modes of motion and two switching conditions. The robot initially moves towards the target until it hits an obstacle. On that occasion, the robot switches to the boundary-following mode of motion and moves along the boundary. It leaves the obstacle boundary and resumes its motion towards the target when a leaving condition, which monitors a globally convergent criterion, holds. In two-dimensions the (bounded) obstacle boundary is a loop. If the leaving condition does not hold and the robot completes a loop around an obstacle, it concludes that the target is unreachable.

However, all existing Bug algorithms plan paths in two-dimensional configuration spaces. Extending these algorithms to three-dimensions is difficult for the following reasons. First, the obstacle boundaries are surfaces, while the robot’s path is a one-dimensional curve. Thus, to conclude target unreachability the robot cannot merely complete a loop around the obstacle. The robot must rather verify that the leaving condition is not satisfied on the entire surface of the obstacle blocking its way to the target. Second, a point robot moving along a three-dimensional obstacle boundary has an infinite number of possible directions for passing around the obstacle, in contrast to only two possibilities in the planar case. Last, the convergence mechanisms used in planar algorithms do not scale up to three-dimensions. Recently, Kutulakos and coworkers studied the problem of three-dimensional sensor-based navigation [12]. They conclude that the robot must explore entire obstacle surfaces to be able to conclude target unreachability, and that the
robot must use a range sensor and maintain a model of the sensed surface to accomplish this task. They also suggest a scheme for 3D path planning which combines a 2D Bug algorithm with a 3D surface-exploration algorithm as follows. The 2D Bug algorithm is activated on a pre-defined plane. When the 2D algorithm fails to find the target, the exploration algorithm is invoked on the obstacle which blocks the way to the target. The 2D navigation is resumed when the exploration algorithm finds a point on the pre-defined plane which satisfies the leaving condition. If no such point is found and the entire surface has been explored, the algorithm concludes that the target is unreachable.

The findings of [12] implies that the reactive behavior of the planar algorithms must be substantially relaxed during the three dimensional exploration stage. Rather, some data structure must be maintained to direct the exploration and to conclude that the entire surface has been explored. Next we describe some known techniques for surface exploration. Rao et al. [20] propose an algorithm for sensor-based model acquisition of general polyhedra. In this algorithm, a robot equipped with an abstract visual sensor incrementally constructs a complete model of the polyhedron by visiting all the vertices. Kutulakos et al. [11] suggest to direct the exploration of smooth surfaces using the visible rim, which is the collection of curves that separate explored patches of the surface from unexplored patches. At each step the robot moves to a point \( v \) on the visible rim and observes some unexplored patch from \( v \). The exploration is completed when the visible rim vanishes. However, the incremental update the visible rim requires a description of all the previously explored patches of the surface, and this representation is not more compact than a complete surface representation. There are also several computer vision papers which address the problem of visual exploration (e.g., [1, 6, 3, 27]). However, these approaches do not guarantee complete exploration for arbitrary objects.

In previous work [9], we presented the TangentBug algorithm which uses range data for making locally optimal decisions in planar environments. TangentBug relies on the fact that in two-dimensional polygonal spaces the shortest path lies on the tangent graph, whose edges are tangent to convex obstacle vertices. But in three-dimensional polyhedral environments the shortest path can pass through any point on the obstacle edges. Therefore it is not possible to directly reduce the search for the shortest path to a graph search. The problem of computing the shortest path in a completely known three-dimensional polyhedral space is NP-hard [2]. The shortest path can be computed in an exponential time \( 2^{\sigma(n)} \), where \( n \) is the number of obstacle edges [21]. However, an \( \epsilon \)-approximate shortest path can be found in a polynomial time of \( O(n^2) \), by selectively breaking obstacle edges into short segments and searching a graph whose nodes consist of sub-edges [4, 19].
In this report we have two objectives. First, we present new results in sensor-based surface exploration and locally shortest path calculation in three-dimensional polyhedral environments. Second, we incorporate these results into a new globally convergent navigation algorithm. We show that a robot with a range sensor can visually explore the entire surface of a polyhedral obstacle by tracing only convex edges which intersect the convex hull of the obstacle. We also show that the vertices of shortest paths in three dimensions lie only on convex edges. Next we define the notion of a locally shortest path, and describe an efficient method for estimating this path. Based on these results, we present a reduced data structure which supports both surface exploration and computation of locally shortest paths. We then present a new Bug-type algorithm, termed 3DBug, which navigates a point robot in three-dimensions using position and range sensors. The 3DBug algorithm uses only convex obstacle edges for surface exploration and locally shortest path calculation. The new algorithm falls within the general framework of the Bug paradigm, since it strives to process the sensory data in the most reactive way possible, without sacrificing the global convergence guarantee. The robot moves between obstacles towards the target in a purely reactive fashion along the locally optimal path, in a way which decreases its distance to the target. The robot switches to the motion mode of surface traversing when it can no longer decrease its distance to the target. During surface traversing, the robot attempts to reach a suitable leave point along a locally shortest path, while simultaneously expanding its knowledge of the obstacle surface based on the range data. If the target is reachable, the robot always finds a leave point and resumes its motion towards the target. Otherwise the robot eventually possesses full knowledge of the obstacle surface, and it determines that the target is unreachable. 3DBug uses 3D range data and plans 3D motion throughout the navigation process. This is in contrast with Ref. [12], where the motion towards the target and the convergence mechanism are restricted to a plane. Thus 3DBug provides a new and more effective Bug-type navigation algorithm for three-dimensional space.

The paper is organized as follows. In the next section we discuss some basic results necessary for sensor-based motion planning in three-dimensions. First we characterize the sensor data and discuss sensor-based surface exploration. We then define the locally shortest path in three-dimensions and describe an efficient method for estimating it. In Section 3 we present the 3DBug algorithm. In Section 4 we discuss the convergence properties of 3DBug. In Section 5 we describe simulation results, showing that in simple environments 3DBug generates paths which often resemble the globally shortest path to the goal. Finally, the concluding section mentions several potential applications of 3DBug.
2 Basic Results in Three Dimensional Navigation

In this section we define the sensor model and characterize its data. Then we discuss some basic results necessary for sensor based motion planning in three-dimensions. We consider a point robot which navigates in a three-dimensional unknown environment populated by stationary polyhedral obstacles. We show that the shortest path in three dimensions is piecewise linear, such that the path vertices lie only on convex obstacle edges. We also show that the robot can visually explore the entire surface of a given polyhedron by tracing the convex obstacle edges which intersect the convex hull of the polyhedron. Last we focus on local information, define the locally shortest path and present an efficient method for estimating it.

2.1 Sensor data

We assume a range sensor with infinite detection range, which provides perfect readings of the distance of the robot from the obstacles within the visible set. This is the three-dimensional star-shaped set centered at the robot current location \( x \). The range sensor maps the visible surfaces of the three-dimensional obstacles into a small sphere centered at \( x \), termed the viewing sphere. The results of this mapping, termed the visibility mapping, have the following characteristic. Every point on the viewing sphere, which determines a viewing direction from \( x \), contains the range to the closest visible obstacle in that direction. (Practical range and vision sensors usually map the visible surfaces onto a plane, using perspective projection. But the planar representation can be readily transformed into our viewing sphere model.) In the following, we assume that the range sensor provides us the 3D coordinates of all visible vertices and edges of the obstacles (Figure 1(a)). An obstacle edge is termed convex if there exists a plane which passes through the edge so that the obstacle locally lies in one half space only (Figure 1(b)). An edge is termed concave if it is not convex. Each visible surface is bounded by edges of two types - either a convex obstacle edge, or an edge generated by occlusion (Figure 1(a)). For computation efficiency, we model each visible surface as a polyhedral two-sided thin wall (or shell) in the real world.

To gain insight into the possible visual arrangements on the viewing sphere, we discuss some relevant results of singularity theory [28]. Singularity theory is concerned with smooth mappings between surfaces, such as the mapping between surfaces in three-dimensions and the two-dimensional viewing sphere. Abrupt changes in the range data on the viewing sphere, called singularities, occur at points which correspond to points on
the obstacle surfaces where the Jacobian of the visibility mapping loses rank. Singularity theory guarantees that there are only three kinds of generic singularities, while all other singularities disappear under small movement of the object or of the direction of projection. We consider only the three stable singularities in our analysis, which are of the following types. The \textit{occluding contour} singularity (also called \textit{fold}) is the only stable curve singularity and it is generated when the viewing direction is perpendicular to the surface normal. On the viewing sphere, occluding contours are curves along which range discontinuity takes place. In a polyhedral environment, the obstacle parts which generate the occluding contours are visible convex edges (Figure 1(b)). The second type of stable singularity is the \textit{T-junction}, which is the transversal superposition of two occluding contours. This singularity appears on the viewing sphere as the intersection point of two occluding curves. The third singularity, called a \textit{cusp}, is a point where a visible occluding contour terminates.

\section{2.2 Sensor Based Surface Exploration}

In this section we show that the entire surface of a given polyhedron can be visually explored while tracing only convex edges. We focus on a single connected surface which lies in the connected component of $\mathbb{R}^3$ which contains the robot. Given a polyhedral obstacle $\mathcal{B}$, we denote the convex hull of $\mathcal{B}$ by $\text{Co}(\mathcal{B})$, and the set of convex obstacle edges which intersect $\text{Co}(\mathcal{B})$ by $\mathcal{S}$. First we show that the shortest path in three-dimensions is piecewise linear, such that the path vertices lie only on convex obstacle edges. Based on this observation, we show that it is possible to move between any two edges in $\mathcal{S}$ along a
collision-free piecewise linear path whose vertices lie on edges in \( \mathcal{S} \). We then show that if the robot traces all the convex edges in \( \mathcal{S} \), every point on the surface of \( \mathcal{B} \) becomes visible to the robot.

To demonstrate the concept of visual exploration by tracing convex edges, consider the following analogous two dimensional example. In two dimensions we consider exploration of a closed polygon by visiting all the convex vertices in the convex hull of the polygon. It can be verified that every point on the boundary of the polygon is visible from some convex vertex which lies in the convex hull. Consider the example shown in Figure 2(a). In this example, the entire boundary of the obstacle \( O_1 \) is visible from the set of convex vertices \( V_1 \ldots V_6 \) contained in the convex hull of \( O_1 \). Moreover, it can be verified that this set is the minimal set of vertices which guarantees coverage, since removing a single vertex can destroy the coverage property. In the example shown in Figure 2(b), removing the vertex \( V \) destroys the coverage. The differences between two-dimensions and three-dimensions should be emphasized at this point. In two dimensions the basic elements are convex vertices, and visiting these vertices is sufficient to guarantee coverage. Moreover, there is a natural order for visiting the convex vertices. The robot can circle around the obstacle in either clockwise or counterclockwise direction, and it concludes that the exploration is complete after completing a loop around the obstacle. In contrast, in three dimensions the basic elements are convex edges, and each of these edges must be entirely traced to guarantee coverage. Moreover, three dimensional exploration requires special data structures and strategies to direct the exploration and to conclude that the entire surface has been explored.

The set \( \mathcal{S} \) of convex obstacle edges which intersect the convex hull \( Co(\mathcal{B}) \) is sufficient for exploration for the following reason. Assuming that the robot is located in \( Co(\mathcal{B}) \), the lines of sight which connect the robot location to other points on the surface necessarily lie in \( Co(\mathcal{B}) \). An occluding face which causes partial occlusion of \( \mathcal{B} \) must block such lines of sight, thus it must lie in \( Co(\mathcal{B}) \). It follows that a convex obstacle edge which bounds an occluding face must lie in \( Co(\mathcal{B}) \). We now focus only on paths which lie in \( Co(\mathcal{B}) \). We denote by \( \mathcal{F}' \) the free space inside \( Co(\mathcal{B}) \). \( \mathcal{F}' \) has polyhedral boundaries since the convex hull of a polyhedron is also a polyhedron. By definition, the surface of \( \mathcal{B} \) lies entirely in \( \mathcal{F}' \), hence it is possible to find a path in \( \mathcal{F}' \) between any two points on the boundary of \( \mathcal{B} \). The following lemma characterizes the shortest path in three dimensions.

\[1\text{In the special case where } \mathcal{B} \text{ defines the boundary of the connected component of } \mathcal{F} \text{ which contains the robot, } \mathcal{F}' = \mathcal{F} \text{ holds.}\]
Figure 2: (a) 2D example of visual boundary exploration. The entire boundary of the obstacle $O_1$ is visible from the convex vertices inside the convex hull of $O_1$, $V_1...V_6$. (b) All the convex vertices are necessary to guarantee boundary coverage in two-dimensions.

**Lemma 2.1 (Shortest paths in three-dimensions)** The shortest path in a three-dimensional polyhedral environment is piecewise linear, such that the path vertices lie only on convex obstacle edges.

**Proof:** It can be verified, using tools like curve variations ([26], pp. 164), that the shortest path in a three-dimensional polyhedral environment is piecewise linear, such that the path vertices lie on obstacle edges. We only show that these vertices lie only on convex obstacle edges. Let $Q = [q_0, q_1, ..., q_n]$ be the shortest path between two points $q_0$ and $q_n$, where $q_i$ for $0 \leq i \leq n$ are vertices on obstacle edges. Assume by contradiction that the shortest path $Q = [q_0, q_1, ..., q_n]$ touches a concave edge at a point $q_i$. Consider the plane $A$ defined by the points $q_{i-1}, q_i$ and $q_{i+1}$. The three dimensional polyhedral obstacles induce polygonal obstacles in the plane $A$, and in particular the point $q_i$ is a concave obstacle vertex in $A$. The path $[q_{i-1}, q_i, q_{i+1}]$ is not locally optimal in $A$, since the locally optimal path in the plane passes only through convex vertices [14]. Hence $Q$ is not the shortest path—a contradiction.

Next we discuss visual characteristics of the set of convex edges $S$.

**Lemma 2.2** For every pair of convex edges in $S$, $E_S$ and $E_F$, there exists a chain of convex edges in $S$, $E_S = E_1, E_2, ..., E_k = E_F$, such that $E_{i+1}$ is visible from $E_i$.

**Proof:** Let us fix two points on the edges $E_S$ and $E_F$, a point $p_S$ on $E_S$, and a point $p_F$ on $E_F$, such that $p_S, p_F \in \text{Co}(B)$. Consider the shortest path from $p_S$ to $p_F$ in $F$.
According to Lemma 2.1, the path $Q = [p_1 = q_1, q_2, ..., q_k = p_F]$ is piecewise linear, and its vertices lie on convex edges in $\mathcal{F}$. Since all the convex edges in $\mathcal{F}$ belong to the set of edges $\mathcal{S}$, each point $q_i$ lies on a convex edge $E_i \in \mathcal{S}$. Since by construction $q_{k+1}$ is visible from $q_k$, the corresponding edge $E_{k+1}$ is visible from $E_k$. Hence the path $Q$ induces a chain of edges in $\mathcal{S}$ such that $E_{k+1}$ is visible from $E_k$.

Lemma 2.2 implies that the set of edges $\mathcal{S}$ can be constructed incrementally. Assume that the robot is initially located within the convex hull $Co(\mathcal{B})$ and on a convex edge $E_i \in \mathcal{S}$. Consider all the shortest paths in $\mathcal{F}$ from start points on $E_i$ to target points on other edges in $\mathcal{S}$. By tracing the edge $E_i$, the robot sees the first segment of each shortest path. Thus the robot can move along the path and see all the convex edges along the path. The next corollary summarizes this fact.

**Corollary 2.3** All the edges in $\mathcal{S}$ become visible to the robot during an exploration process in which the robot traces every convex edge it detects.

Next we show that the entire surface of $\mathcal{B}$ can be explored by tracing the edges of $\mathcal{S}$. It is possible to prove that every point on the surface is visible from some convex edge in $\mathcal{S}$ using similar arguments to those used in Corollary 2.3. We will prove a stronger argument, that it is possible to explore the entire surface from a finite number of points. We first show that for every convex polygon $P$ which lies on the surface of $\mathcal{B}$, there is an observation point $o$ which lies on an edge $E \in \mathcal{S}$ such that the entire polygon $P$ is visible from $o$.

**Lemma 2.4** For every convex polygon $P$ which lies on the surface of $\mathcal{B}$, there is an observation point $o$ which lies on a convex edge $E \in \mathcal{S}$ such that the entire polygon $P$ is visible from $o$.

**Proof:** Given a convex polygon $P$ which lies on the surface of $\mathcal{B}$, we denote as $W$ the set of points in $\mathcal{S}$ from which $P$ is visible, and denote as $dist(w, P)$ the distance from a point $w$ to $P$ i.e., the minimal distance between $w$ and points in $P$. We choose the observation point $o$ such that it has the minimal distance to $P$ among the points in $W$, $d(o, P) = \min_{w \in W} dist(w, P)$. First we show that $W$ is not empty. Consider a point $p_1$ on the surface of $\mathcal{B}$ from which the polygon $P$ is not visible, and a point $p_2 \in P$. Construct the shortest path from $p_1$ to $p_2$ in $\mathcal{F}$, $Q = [p_1 = q_1, q_2, ..., q_k = p_F]$. The last vertex before $p_2$, $q_{k-1}$, lies on an edge of $\mathcal{S}$, and $P$ is visible from $q_{k-1}$, thus $W$ is not empty.

Next we show that the polygon $P$ cannot be partially occluded when viewed from $o$. Let us first consider the case in which $o$ does not lie on the plane $A$ which contains $P$. 

...
Figure 3: The distance \( d(x, P) \) from a point \( x \) to the convex polygon \( P \) monotonically decreases along the segment \([o, p]\).

Assume by contradiction that \( P \) is occluded by an occluding edge \( E_h \). The edge \( E_h \) must be a convex edge which intersects the convex hull of \( B \), thus \( E_h \in S \) holds. Consider a line segment \([o, h, p]\) such that \( h \) lies on \( E_h \) and \( p \in P \), and consider a point \( x \) on the segment \( x = ao + (1 - a)p \). We will show that \( dist(x, P) \) strictly decreases along the segment from \( o \) to \( p \). The distance \( d(x, P) \) can be written as \( \sqrt{d_Z(x)^2 + d_A(x)^2} \), where \( d_Z(x) \) is the (vertical) distance from \( x \) to \( A \), and \( d_A(x) \) is the (horizontal) distance from the projection of \( x \) on the plane \( A \) to \( P \) (Figure 3). The distance \( d_Z \) is a convex function, which satisfies \( d(x, P) \leq ad(x_o, P) + (1 - a)d(x_1, P) \). Considering the distance \( d_A(x) \), we get \( d_A(x) \leq ad_A(o) + (1 - a)d_A(p) \). Given that \( d_A(p) = 0 \), it follows that \( d_A(x) \leq d(o, P) \). Thus the sum \( \sqrt{d_Z(x)^2 + d_A(x)^2} \) decreases monotonically. It follows that the occluding point \( h \) is closer to \( P \) than \( o \), \( d(h, P) < d(o, P) \), and that contradict the choice of \( o \) as the point with minimal distance to \( P \). For the second case, in which \( o \) lies on the plane \( A \) which contains \( P \), it can be verified that \( dist(x, P) \) strictly decreases. □

**Theorem 1** The entire surface of a polyhedron \( B \) is visible from a finite number of observation points on the convex edges which intersect the convex hull of \( B \).

**Proof:** Consider a partition of the surface of \( B \) to a finite number of convex polygons. Based on Lemma 2.4, it is possible to find, for each convex polygon \( P \), an observation point \( o \) from which \( P \) is visible. Hence the entire surface of a polyhedron \( B \) is visible from a finite number of observation points. □

Theorem 1 implies that by tracing all the convex edges in \( S \), which intersect the convex hull of \( B \), a robot with a range sensor can visually explore the entire surface of \( B \). Moreover, the
set $S$ is the minimal set of edges which guarantees complete exploration, since discarding a single edge from $S$ can destroy the covering property (see 2D example in figure 2(b)). Hence the set $S$ supports both the calculation of the shortest paths between points on the surface of $B$, and the exploration of the surface.

The CEG data structure: We now define a data structure termed the *Convex Edges Graph*, CEG, in which each convex obstacle edge is a node. The current robot location $x$ and the target location $T$ are added as special nodes, to allow calculation of paths from $x$ to $T$. To calculate paths between points on the surface of $B$ the graph must be connected, hence we define CEG edges\(^2\) that maintain graph connectivity. The CEG edges belong to one of the following three types: (1) point edges, which are shared endpoints of the nodes i.e., obstacle vertices in which two convex edges meet, (2) linking edges, which connect disjoint subsets of nodes, and (3) virtual edges, which connect $x$ and $T$ to other nodes. For each edge we store the coordinates of its endpoints and the edge weight. The detailed construction of the CEG is described in section 3.4.

Next we define the path length on the CEG. Recall that CEG nodes represent obstacle edges with non-negligible lengths, and that each CEG edge is defined by the 3D coordinates of its endpoints. A path in the CEG is defined with respect to start and final points which lie on CEG nodes. (We usually calculate shortest paths from $x$ to $T$, which are special CEG nodes). Two adjacent edges along the path, which enter and exit the same CEG node, may have different 3D coordinates (Figure 11). Let $L_V$ denote the sum of Euclidean distances between endpoints of adjacent edges along the path. Let $L_E$ denote the accumulated edge length along the path. The total path length $L$ is defined as a sum, $L = L_E + L_V$.

The CEG is compact and easy to maintain relative to other data structures which were proposed for sensor based surface exploration. First, the set $S$ of convex edges which intersect the convex hull of $B$ is minimal in the following sense. Discarding a single edge from $S$ can destroy the covering property i.e., there may be a point on the surface of $B$ which is not visible from an edge in $S$ (see 2D example in figure 2(b)). Since concave edges are not stored in the CEG, it is typically more compact than the complete polyhedral model, suggested in [20], in which all the vertices are stored. Moreover, the advantage of the CEG becomes more evident when the data structure must support shortest path calculations. In this case both the complete model and the CEG must consider all the objects which lie in the convex hull of $B$$^3$. Hence the complete model would contain all

\(^{2}\)Note that the word “edge” is used with two meaning: the CEG nodes are convex obstacle edges while the CEG edges are abstract entities.

\(^{3}\)The globally shortest path can pass through convex edges outside the convex hull of $B$. Such edges
the obstacle features which lie in the convex hull, while the CEG contains only the convex edges in the convex hull. Last, the CEG is easy to update and maintain, relative to the visible rim which was suggested in [11], due to the following reason. Updating the visible rim requires a description of the previously explored part of the surface, while the CEG does not require any representation of visible surfaces.

2.3 Locally Shortest Path in Three-Dimensions

We define the locally shortest path as the shortest collision-free path from the robot current location $x$ to the target $T$, based only on the currently observable obstacles. We wish to use the locally shortest path during the reactive mode of motion towards the target, and this section presents an efficient method for approximating the locally shortest path. Recall that each sensed surface is modeled as a polyhedral two-sided thin wall. If the target is not visible from $x$, there is a blocking obstacle between the robot and the target. In this case the line segment $[x, T]$ crosses the blocking obstacle, and we refer to the visible surface of the blocking obstacle as the blocking surface. The blocking surface is bounded by a piecewise linear curve, termed the blocking contour (Figure 4(a)). By construction, the blocking contour lies on the blocking obstacle, and its edges are of the following two types. Occluding edges are generators of occluding contours, and they are typically silhouette edges of the blocking obstacle. Occluded edges are generated from occlusion by some other obstacle. Each occluded edge of the blocking contour has a corresponding occluding edge, which is a convex obstacle edge that partially occludes the blocking obstacle (Figure 4(b)).

The computational effort necessary for generating the locally shortest path is an im-
important issue in the design of an effective algorithm. As stated in the introduction, the
globally shortest path is difficult to compute in three-dimensions. In the following, we show
that the search for the locally shortest path can be restricted to paths passing through
the blocking contour. This property is the basis for the elegant straight forward estimation
of the locally shortest path that we develop below. Note that the globally shortest
path does not satisfy this property. First we prove that the locally shortest path passes
through the blocking contour for two-dimensions, then extend it for three-dimensions. In
two-dimensions, we assume polygonal obstacles and model the sensed obstacles as thin
one-dimensional walls.

**Proposition 2.5** Assume a planar polygonal environment, with a blocking obstacle be-
tween the robot location $x$ and the target $T$. Then the shortest path from $x$ to $T$, con-
sidering only the currently visible obstacles, must pass through an endpoint of the
blocking obstacle.

**Proof:** It is known that the shortest collision-free path from $x$ to $T$ consists of straight
line segments which pass through convex obstacle vertices. Let $\gamma_1$ be the shortest path to
$T$ which passes through the right endpoint of the blocking obstacle, denoted $v_0$. Suppose
there exists a different path from $x$ to $T$, $\gamma_2$, which is the shortest among all paths
circumventing the blocking obstacle from the right (Figure 5). Let $Poly_1$ and $Poly_2$ be
the polygons whose edges are the line segment $[x,T]$ and the polygonal paths $\gamma_1$ and $\gamma_2$,
respectively.

We now show that the path $\gamma_2$ does not cross $\gamma_1$, although it may partially overlap
$\gamma_1$. First, it can be verified that $\gamma_2$ does not cross $\gamma_1$ in the segment $[x,v_0]$. If $\gamma_2$ crosses
$\gamma_1$ at a point $c$ after the segment $[x,v_0]$, there would be two possible sub-paths. One

Figure 5: The shortest path from $x$ to $T$, among all paths circumventing the blocking
obstacle from the right, must pass through the endpoint $v_0$ of the blocking obstacle.
Figure 6: The shortest path from $x$ to $T$ passes through a point of the blocking contour.

from $c$ to $T$ along $\gamma_1$ and the other from $c$ to $T$ along $\gamma_2$. However, both $\gamma_1$ and $\gamma_2$ must follow the shorter sub-path, since both paths are defined as shortest paths, and therefore cannot be shortened. Thus $\gamma_2$ never crosses $\gamma_1$. Since $\gamma_2$ circumvents the blocking obstacle from the right and does not cross $\gamma_1$, the polygon $Poly_2$ strictly includes $Poly_1$. We now show that $\gamma_2$ cannot possibly be shorter than $\gamma_1$. For every line segment $[y_i, y_{i+1}]$ of $\gamma_1$, consider the two rays perpendicular to the segment and directed outward with respect to $Poly_1$, e.g., the segment $[x, v_0]$ in figure 5. Let $[z_i, z_{i+1}]$ be the portion of $\gamma_2$ bounded by these two rays e.g., the segment $[x, z_1]$ in figure 5. Clearly $[z_i, z_{i+1}]$ is not shorter than $[y_i, y_{i+1}]$. Moreover, it can be verified that $Poly_1$ is convex, since $\gamma_1$ passes through convex obstacle vertices of a single visible obstacle. Hence portions $[z_i, z_{i+1}]$ of $\gamma_2$ corresponding to different segments $[y_i, y_{i+1}]$ of $\gamma_1$ are disjoint. It follows that $\gamma_2$ must be longer than $\gamma_1$. (If the two paths have identical length, it can be verified that the paths are identical.) Thus the shortest path to $T$ must pass through one of the two endpoints of the blocking obstacle.

Corollary 2.6 Assume a polyhedral three-dimensional environment, with a blocking obstacle between the robot location $x$ and the target $T$. Then the shortest path from $x$ to $T$, considering only the currently visible obstacles, must pass through a point of the blocking contour (Figure 6).

Proof: Suppose the shortest path from $x$ to $T$ does not pass through a point of the blocking contour. Without loss of generality, assume that the path consists of two line segments, $[x, y]$ and $[y, T]$, where $y$ does not lie on the blocking contour. Let $A$ denote the plane defined by the three points $x, y, T$. Since the line $[x, T]$ crosses the blocking surface, this surface intersects the plane $A$ along a curve $C$, which forms a two-dimensional blocking obstacle in $A$ (Figure 6). According to Proposition 2.5, the shortest path from $x$ to $T$ in
Figure 7: The blocking-contour graph of (a) a convex obstacle (b) a concave obstacle.

the plane $A$ must pass through an endpoint of $C$, which belongs to the blocking contour.

Now we discuss methods for computing the locally shortest path. It is possible to run Papadimitriou’s $\epsilon$-optimal algorithm [19] on the thin-wall model of the currently observable obstacles. However, this algorithm is computationally expensive. We now present an efficient method for estimating the locally shortest path. The resulting estimate will be called the blocking-contour path. For each point $y$ on the blocking contour, consider a path consisting of two line segments: the visible part $[x, y]$ and the (optimistically) expected part $[y, T]$. The length of this path is $L_{\text{block}}(y) = \|x - y\| + \|y - T\|$. For each line segment $l_i$ of the blocking contour, we compute the point $v_i$ which minimizes $L_{\text{block}}(y)$. This computation can be done in constant time per line segment $l_i$. Then we construct a local graph, called the blocking-contour graph, consisting of edges from $x$ to each $v_i$, and (optimistic) edges from each $v_i$ to $T$, as shown in Figure 7. The blocking-contour path is the shortest path on the blocking-contour graph, and it can be found in time linear in the number of line segments in the blocking contour.

We may ask, what is the relation of the blocking-contour path to the exact locally shortest path? If the blocking obstacle is convex, the blocking-contour path becomes identical to the locally shortest path, for the following reason. If the line segment $[y, T]$ intersects the blocking surface at some visible point, the blocking obstacle must have a visible concavity. Hence if the obstacle is convex, the line $[y, T]$ does not intersect the blocking surface and the blocking-contour path is precisely the locally shortest path. However, in general $\|y - T\|$ is merely an optimistic estimate of the path length from $y$ to $T$. Note that to compute the blocking-contour path, knowledge of the blocking contour is sufficient, and it is not necessary to construct a full polyhedral model of the blocking surface. This property simplifies the implementation of the method, and makes it useful for navigation among more general three-dimensional shapes.
3 The 3DBug Algorithm

The 3DBug algorithm navigates a point robot in a three-dimensional unknown environment populated by stationary polyhedral obstacles. The sensory information available to the robot during the navigation consists of the robot current position $x$, and range data in the form of distance from $x$ to every obstacle point within the visible set. First we describe the global structure of the algorithm and then discuss its detailed operation.

3.1 Algorithm Description

The 3DBug algorithm uses two basic motion-modes: motion towards the target and obstacle-boundary traversing. During motion towards the target the robot moves along the *locally shortest path* based on the currently observable obstacles. At each step of this motion mode the robot chooses a *focus point* and moves to this point, without performing any sensing or replanning. When the robot reaches the focus point it senses the environment and chooses the next focus point. While the focus point can be computed continuously during the robot motion, our experience shows that computation of the focus point at discrete steps reduces the computation time without significantly sacrificing the quality of the resulting path. Let $w$ be a point in the free space, and let the function $d(w, T)$ be the Euclidean distance of $w$ to the target $T$. The robot keeps moving towards the target until it becomes trapped in the basin of attraction of a local minimum of $d(w, T)$. The appearance of a local minimum is always associated with the presence of an obstacle which blocks the direct path from the robot location to the target, and at this point the robot switches to traversing the boundary of the blocking obstacle.

During the boundary-traversing mode, the robot searches for a suitable leave point on the boundary from which it can resume its motion towards the target, while simultaneously expanding its knowledge of the obstacle’s boundary. The algorithm uses the *Convex Edges Graph*, CEG, to accumulate information about the followed obstacle, and the construction of the CEG is described in section 3.4. As discussed above, the CEG is the minimal data-structure suitable for the two required tasks—finding locally shortest paths towards candidate leave points and exploring the surface of the obstacle being followed.

At each step during the boundary-traversing mode, the robot calculates the shortest path to the target based on the current CEG, and chooses the next focus point $F$ on this path as follows. Recall that the shortest path is piecewise linear with vertices on convex obstacle edges. The robot chooses the next focus point at the last path-vertex which lies on an obstacle edge. The robot then moves to $F$ while performing local shortcuts relative to the CEG-based shortest path, using the following procedure. When $F$ is visible to the
robot, it moves directly towards \( F \). Otherwise it defines an intermediate point, \( v \), at the furthest currently visible point along the CEG-based shortest path, moves to \( v \) and upon arriving to \( v \) it senses the environment and chooses the next intermediate point. The robot repeats this procedure until it reaches \( F \). Upon reaching \( F \) the robot traces a portion of the convex edge on which \( F \) is located while continuously sensing the environment. During this tracing operation the robot updates the CEG according to the sensed range data. We show in the ensuing analysis that the accumulative effect of these tracing operations is a complete surface exploration, as required in order to guarantee successful navigation. During each tracing operation the robot also continuously records the closest point to the target observed so far on the surface of the followed obstacle. This point is denoted \( p_{\min} \), and the robot additionally records the distance of \( p_{\min} \) to the target, denoted \( d_{\min}(T) \).

After updating the CEG, the robot tests the leaving condition in the following way. It inspects \( v_{\text{leave}} \), which is the closest point to the target along the visible portion of the line segment \([x,T]\), where \( x \) is the current robot location. The robot leaves the obstacle boundary when \( d(v_{\text{leave}},T) < d_{\min}(T) \). After leaving the obstacle, the robot performs a transition phase before it resumes its motion towards the target. In the transition phase the robot moves directly towards \( v_{\text{leave}} \) until it reaches a point \( z \) where \( d(z,T) < d_{\min}(T) \). At this point the robot resumes its motion towards the target.

Last we describe how the robot detects that the target is unreachable. The robot determines that it has completed the exploration of an obstacle surface after it had traced all the convex obstacle edges which exist in the CEG. When an entire obstacle surface has been explored without finding a suitable leave point, the robot performs the following final target-reachability test. The robot moves to the closest point to the target on the obstacle surface, \( p_{\min} \), and checks the leaving condition from there. If the leaving condition is not satisfied, the target is necessarily unreachable and the robot halts its motion. A summary of the algorithm now follows.

1. Move towards \( T \) along the \textit{locally shortest path}, until one of the following events occurs:
   - The target is \textbf{reached}. Stop.
   - A local minimum is detected. Go to step 2.

2. Traverse the obstacle’s surface, searching for a suitable leave point, while updating the CEG and recording \( d_{\min}(T) \), until one of the following events occurs:
   - The target is \textbf{reached}. Stop.
   - The leaving condition holds: \( \exists v_{\text{leave}} \) s.t. \( d(v_{\text{leave}},T) < d_{\min}(T) \).
     Go to step 4.
The entire surface was sensed. Go to step 3.

3. Perform the final test. Go to the closest point to $T$ on the surface.

   If the leaving condition holds at the closest point, go to step 4.

   Otherwise the target is **unreachable.** Stop.

4. Perform the transition phase. Move directly towards $v_{leave}$ until reaching a point $z$

   where $d(z, T) < d_{\text{min}}(T)$. Go to step 1.

### 3.2 Motion Towards the Target

During motion towards the target, the robot moves between successive focus points, denoted $F_i$, without sensing or replanning. When the robot arrives to a focus point it senses the environment and chooses the next focus point along the locally shortest path to the target, based on the currently sensed obstacles. If the target is visible to the robot, the shortest path leads directly towards it. Otherwise, the locally shortest path passes through the blocking contour (Corollary 2.6). The blocking contour is computed directly from the range data, as the locus of range discontinuities on the boundary of the blocking surface.

See section 5 for examples of this computation in several simulated environments.

In order to guarantee convergence to the target, we wish to ensure that the distance of the robot to the target decreases monotonically between successive steps. To achieve this objective, the algorithm computes the locally shortest path based only on the points $y$ of the blocking contour satisfying $d(y, T) \leq d(x, T)$, where $x$ is the current robot location. This subset of the blocking contour is termed the **feasible sub-contour** (Figure 8(a)). In other words, the feasible sub-contour is the portion of the blocking contour contained in the ball of radius $d(x, T)$ centered at $T$. (Points $e$ satisfying $d(e, T) = d(x, T)$ are later excluded from being focus points.) Once the feasible sub-contour is constructed, the algorithm constructs the blocking-contour graph based on the feasible sub-contour and the target node, and searches this graph for the shortest path to $T$.

Once the locally shortest path is computed, the next focus point $F$ is chosen on this path as follows. Let $G$ be the point on the feasible sub-contour through which the locally shortest path passes. (It can be verified that $G$ is unique.) If $G$ lies on a convex edge of the blocking obstacle, $F$ is set to $G$ (Figure 8(a)). If $G$ lies on an edge generated from occlusion, $F$ is chosen on the occluding edge, at the point where the line segment $[x, G]$ crosses the occluding edge (Figure 8(b)). The reason for this choice is as follows. The globally shortest path never passes through occluded edges of the feasible sub-contour. Since we wish to achieve local decisions that resemble the globally optimal ones, we locate $F$ on the occluding edge.
Figure 8: Examples of motion towards the target. (a) From \( x = S \), the locally shortest path passes through the focus point \( F_1 \). (b) At \( x = F_1 \), two edges of the blocking contour result from occlusion. The point \( G \) lies on an occluded edge, hence the next focus point, \( F_2 \), is set on the occluding edge. (c) From \( x = F_2 \), the locally shortest path passes through \( F_3 \), from which the target can be reached directly.

In order to guarantee that each motion-towards-the-target segment would have a finite length, we must exclude cases in which the decrease in \( d(x, T) \) between successive focus points is zero or infinitesimally small. The location of the focus point \( F \) is therefore modified using the following post-processing step. Let \( E \) denote the convex obstacle edge on which \( F \) was chosen, and let \( e \) be a point on \( E \) which satisfies \( d(e, T) = d(x, T) \). (There may be at most two such points.) Then \( F \) is located too close to \( e \) when \( d(e, F) < \delta \) holds, where \( \delta \) is a small positive parameter whose choice is discussed below. In this case the focus point \( F \) is shifted along the edge such that \( d(F, T) \) decreases, until either \( d(e, F) > \delta \) holds true or \( d(F, T) \) is minimal along the edge. In other words, when \( F \) is located too close to the sphere of radius \( d(x, T) \) centered at \( T \), \( F \) is shifted into the interior of the sphere. As shown in the ensuing analysis, the accumulative path length of the motion-towards-the-target segments is finite when this post-processing step is used.

We now describe our criterion for setting the parameter \( \delta \). Shifting the focus point from its original location along the blocking-contour path, from \( F \) to \( F^* \), increases the expected path length to the target, i.e. \( L(F) < L(F^*) \). We now derive the relation between \( \delta \) and \( L(F^*) \). In figure 9, the robot is located at \( x \) and the focus point \( F \) lies on a convex obstacle edge \( E \). The robot computes the angles \( \alpha \) and \( \beta \), which lie on the planes defined by the points \([F, F^*, x]\) and \([F, F^*, T]\), respectively. The ratio between the lengths of the modified path, \( L(F^*) \), and the blocking-contour path, \( L(F) \), is: \( L(F^*) / L(F) = (\|x - F^*\| + \|F^* - T\|) / (\|x - F\| + \|F - T\|) \). Using the cosine law, the length of the line segments \([x, F^*]\) and \([F^*, T]\) is: \( \|x - F^*\| = (\delta^2 + \|x - F\|^2 - 2\delta \|x - F\|)^{1/2} \), \( \|F^* - T\| = (\delta^2 + \|F - T\|^2 - 2\delta \|F - T\|)^{1/2} \). Hence the value of \( \delta \) can be chosen according
Figure 9: By modifying the location of the focus point and shifting it from $F$ to $F^*$, such that $d(F, F^*) = \delta$, the expected path length from $x$ to $T$ increases.

to the criterion $L(F^*)/L(F) < \rho$, where $\rho$ is a user specified increase ratio in the expected path length to the target. However, $\delta$ must always be larger than some minimal value $\delta_0 > 0$.

The robot terminates the motion towards the target and switches to the boundary-traversing mode after detecting that it is trapped in the basin of attraction of a local minimum of the function $d(w, T)$. The sensor-based termination condition is that the feasible sub-contour becoming empty, and it is shown in the ensuing analysis that this event is always associated with the presence of a local minimum. Note that the switch may occur even at the start point $S$, as illustrated in Figure 10.

### 3.3 Traversing an Obstacle Surface

This motion mode has two objectives—to find suitable leave points and to explore the obstacle surface. Let $P$ denote the point where the robot switches to boundary-traversing mode. It can be verified that the local minimum of $d(w, T)$ which terminated the motion towards the target is visible from $P$, and lies on the surface of the obstacle which blocks the way from $P$ to the target. The robot traverses the surface of this obstacle until either a leaving condition is satisfied or the entire obstacle surface is explored. During this motion the robot attempts to reach a suitable leave point along a locally shortest path, while simultaneously expanding its knowledge of the followed obstacle based on the range data.

Upon starting a new boundary traversing segment, the robot performs the following two initial steps. First the robot moves into the convex hull of the blocking obstacle as follows. The robot chooses a focus point $F_0$ as the closest point to the target on the blocking contour, and moves directly to $F_0$. Since the blocking contour lies on the followed
Figure 10: An example of boundary traversing. The robot switches to boundary traversing at $S$, since $d(S,T) < d(y,T)$ holds for every point $y$ on the blocking contour. The robot moves to $F_1$, then traces an edge-segment until it reaches $E_1$. Eventually, the robot reaches $E_3$, from which the target is visible and the leaving condition holds.

obstacle, $F_0$ necessarily lies inside the convex hull of the blocking obstacle. Upon arriving to $F_0$, the robot senses the environment and generates the initial CEG of the followed obstacle.

At each step after the initial one, the robot computes the shortest path to the target according to the current CEG. Before explaining how the robot moves along this path, we informally describe the search and exploration strategy used during boundary traversing. This strategy is implicitly determined by the weights in the CEG, which are described in detail in the next section. Initially the robot focuses mainly on finding suitable leave points which would allow it to resume its motion towards the target. To achieve this task the robot moves to unvisited convex obstacle edges which lie on short paths to the target. It makes sense to move to unvisited edges, since the leaving condition did not hold on the previously visited edges and the leaving condition typically holds or does not hold for entire edges. If the robot is unable to leave the followed obstacle after visiting all the convex obstacle edges, it shifts its focus to the exploration of the entire obstacle surface. Complete coverage of the obstacle surface is necessary both for finding a leave point in complex scenarios and for concluding target unreachability. During this stage the robot traces all the convex obstacle edges (which are nodes of the CEG), and terminates its motion when all the edges corresponding to CEG nodes have been traced.

We now describe how the robot moves during boundary traversing. Given the CEG-
based shortest path to the target, the robot chooses the next focus point $F$ as the last path-vertex which lies on an obstacle edge. (The shortest path is piecewise linear with vertices on convex obstacle edges.) The robot then moves to $F$ by repeatedly performing the following procedure. Let $\gamma$ be the CEG-based shortest path to the target. The robot chooses the furthest point $v$ along $\gamma$ which is visible from its current location $x$, and moves directly towards $v$ without performing any sensing. After reaching $v$, the robot senses the environment, and repeats the same procedure of moving to the furthest visible node along $\gamma$, until it reaches the focus point $F$. Note that this method generates shortcuts relative to the shortest path in the CEG (Figure 11(b)).

Next we describe the tracing of convex obstacle edges. Let $E$ denote the convex obstacle edge on which the focus point $F$ is located. After reaching $F$, the robot traces a non-traced portion of $E$ while continuously sensing the environment and updating the CEG (Figure 10). During the tracing operation the robot continuously computes the closest point to the target observed so far on the followed obstacle, $p_{\text{min}}$, and records its distance to the target in the variable $d_{\text{min}}(T)^4$. The robot also records a CEG node from which $p_{\text{min}}$ is visible. The length of each traced edge-segment is set to $\lambda\|E\|$, where $0 < \lambda \leq 1$ is a user specified parameter called the tracing factor, and $\|E\|$ is the length of the edge. In the simulations section we discuss one particular scheme for choosing $\lambda$ which allows efficient surface exploration. We are currently developing a scheme for automatically increasing $\lambda$ from 0 to 1 during a boundary traversal segment, according to the principle that small values of $\lambda$ emphasize the search for suitable leave points, while large values of $\lambda$ emphasize surface exploration.

After updating the CEG, the robot tests the leaving condition as follows. The robot inspects $v_{\text{leave}}$, the closest point to the target along the visible portion of the segment $[x, T]$, where $x$ is the current robot location. The leaving condition is satisfied when $d(v_{\text{leave}}, T) < d_{\text{min}}(T)$. Note that different versions of the leaving condition, which may allow the robot to leave the followed obstacle earlier, can be incorporated into the algorithm. In particular, it is possible to test the leaving condition continuously during the tracing of an edge segment, not only at the end point of the traced segment. After the entire surface has been explored without finding a leave point, the robot performs the following final target-reachability test. The robot moves to the closest point to the target on the surface, $p_{\text{min}}$, and checks the leaving condition from there. If the leaving condition is not satisfied from that point, the target is unreachable. This final test is necessary since the leaving condition is tested

---

4To compute $p_{\text{min}}$, the robot applies the following procedure for every polygonal patch $P$ on the visible surface. First, the robot computes the closest point to $T$, denoted $y$, on the plane which contains $P$. If $y$ lies inside $P$, then $p_{\text{min}} = y$, otherwise $p_{\text{min}}$ is the closest point to $T$ on the boundary of $P$. 

only at discrete points on convex obstacle edges, and in general these points do not suffice to conclusively determine target unreachable.

Finally, after leaving the obstacle, the robot performs a transition phase before it resumes its motion towards the target. In this phase the robot moves directly towards $v_{leave}$ until it reaches a point $z$ where $d(z, T) < d_{min}(T)$. As discussed below, the transition phase ensures that the distance to the target of the local-minima which terminate the motion-towards-the-target segments decrease monotonically.

### 3.4 Construction of the Convex Edges Graph

The Convex Edges Graph, CEG, was defined in section 2.2. This is a compact data structure which supports the two tasks of computing locally shortest paths on an obstacle surface and exploring the surface of the obstacle. Here we give the finer details of the CEG structure, then describe the weight assignment on the CEG edges, and finally describe the on-line construction of the CEG during boundary-traversing mode.

#### 3.4.1 The CEG Structure

The CEG consists of nodes which represent convex obstacle edges, and edges which represent paths between the respective convex obstacle edges. We now define in detail the CEG structure. Let $B$ be the obstacle whose surface is being followed, and let $Co(B)$ denote the convex hull of $B$. The CEG nodes represent convex obstacle edges which lie in $Co(B)$ and have been seen by the robot during boundary traversing. (The current robot location $x$ and the target location $T$ are added as special nodes, to allow calculation of paths from $x$ to $T$.) For every CEG node, we store the 3D coordinates of the endpoints of its corresponding obstacle edge and the endpoints of the traced portion of this edge. Note that the visible portion of an obstacle edge corresponding to a CEG node only extends during exploration. Moreover, two CEG nodes may merge into a single node during the exploration. Typically, the traced portion of each CEG node is a single continuous subsegment. However, when two nodes are merged, their traced portions may be non-connected since the traced portion is only a subsegment of the edge-segment which has been seen by the robot.

Next we describe the CEG edges. There are three types of CEG edges, and for each edge we store the 3D coordinates of its endpoints and the edge weight. Type 1 edges, termed point edges, connect obstacle edges which share a vertex. The weight of such edges is always zero (Figure 11(a)). Type 2 edges, termed linking edges, guarantee that the CEG is connected by the end of each boundary-traversing step. The linking edges are
chosen when the robot is still, after the CEG nodes were updated and point edges were added. These edges are selected based on the currently observable obstacles, according to the following three criteria. Linking edges connect disjoint subsets of CEG nodes, they do not intersect any obstacle (i.e., they are visibility edges), and they have minimal length among the edges which satisfy the two previous requirements. It is always possible to make the CEG a connected graph by adding such linking edges, since it is possible to move between any two CEG nodes along a chain of convex obstacle edges which are visible from each other (Lemma 2.2). The weight of each linking edge is set as the Euclidean distance between its endpoints. Note that the endpoints of a linking edge may lie in the interior of obstacle edges, as depicted in Figure 11(a). Type 3 edges, termed virtual edges, connect the points $x$ and $T$ to the CEG nodes. The robot location $x$ is connected via a zero-weight virtual edge to the CEG node on which the robot is located. The weights of the virtual edges which connect $T$ to the CEG nodes determine to a large degree the robot’s path during boundary traversing. These weights are described in the next section.

Note that the CEG does not include most of the edges which indicate visibility relations between convex obstacle edges. Explicit representation of these edges can be avoided since the robot uses its sensors to find visibility relations between currently observable obstacle edges. In particular, during boundary traversing the robot moves towards the focus point while performing local shortcuts relative to the CEG-based shortest path (Figure 11(b)).

3.4.2 Weight Assignment on the Virtual Edges

The weights of the virtual edges which connect $T$ to the CEG nodes determine to a large degree the robot’s path during boundary traversing, since the robot chooses the next focus point along the shortest path on the CEG. These weights are set and modified during the exploration according to the following two criteria. Initially, the robot moves to unvisited convex obstacle edges along short paths to the target, since in this stage suitable leave points are most likely to be found merely by visiting these edges. After visiting all the convex edges detected so far without finding a suitable leave point, the robot should efficiently trace all the convex edges in order to determine target unreachability.

For each convex obstacle edge $E$, the virtual edge connecting it to $T$ emerges from the optimal point $V_{opt}$ which is defined as follows. If the edge $E$ has not been visited by the robot yet (only observed), its optimal point is the closest point to $T$ along the visible portion of the edge. The weight of the virtual edge $[V_{opt}, T]$ is set to the length of the shortest path from $V_{opt}$ to $T$, $L(V_{opt})$, considering only the currently visible obstacles. (The weight of a virtual edge is updated only when the visible portion of its CEG node is
Figure 11: (a) The CEG components. Two disjoint sets of CEG nodes are visible from the robot location at $F_1$, and a linking edge connects them. (b) After reaching $F_1$, the robot traces a fixed-length segment of the edge until it reaches the point $e_1$. From $e_1$ the robot calculates the shortest path to $T$ and chooses $F_2$ as the next focus point. Note that in the next boundary-traversing step, the robot will move from $e_1$ directly towards $F_2$, thus performing a shortcut relative to the CEG-based shortest path.

An efficient method for estimating the shortest path length $L(V_{opt})$ is presented in section 2.3. If the robot has already visited the edge $E$, the optimal point is chosen in the following way. Let $O_E$ denote the set of endpoints of the traced subsegments on $E$ which are next to non-traced portions of the edge. Then the optimal point $V_{opt}$ has the minimal distance to $T$ among all the points in $O_E$. (In the example presented in figure 11(b), $O_E$ of the edge containing the robot consists of the endpoints $F_1$ and $e_1$, and $V_{opt}$ is set to $F_1$.) In this case the weight of the edge is set to $L(V_{opt}) + c_1$, where $c_1$ is a constant whose setting is discussed below. If the entire edge $E$ has been traced, $O_E$ is empty and consequently there is no optimal point on $E$, and no virtual edge to $T$.

Next we explain why this assignment of weights to virtual edges generates the desired search strategy. First, we wish to ensure that as long as there are unvisited convex obstacle edges, the focus point is chosen on one of these edges. To achieve that, the constant $c_1$ is set as follows. Let $L_{max}(B, T)$ be the maximal distance between $B$ and $T$. Then $c_1 > L_{max}(B, T) + \max_{x_1, x_2 \in B} L(x_1, x_2)$, where $\max_{x_1, x_2 \in B} L(x_1, x_2)$ is the maximum path length over the collection of all CEG-based paths between pairs of points on the obstacle edges. Using this weight assignment, $c_1$ is the maximal path length possible on the CEG. It follows that paths through visited edges are always longer than paths through unvisited edges, and the focus point is consequently chosen on unvisited edges. After visiting all the convex obstacle edges detected so far, all the virtual edges to $T$ have weights larger
Figure 12: The convex vertex $V$, which lies within the convex hull of $O_1$, never causes partial occlusion of $O_1$. In contrast, the vertex $Y$ causes partial occlusion from the current robot location at $x$.

than $c_1$. At this stage of the exploration the robot traces edges in the following way. The robot starts tracing from the current focus point which is, by construction, the optimal point of some convex obstacle edge $E$. If the tracing is terminated next to a non-traced portion of $E$, the robot location $x$ is likely to be chosen as the new optimal point of $E$. In this case the next focus point will typically be chosen at the current robot location $x$, and the robot will continue to trace the same edge. Hence the tracing process is efficient. The exploration is completed when no paths to $T$ exist in the CEG. This situation happens only when there are no virtual edges to $T$, and by construction, it implies that all the convex obstacle edges have been traced.

### 3.4.3 On-Line Construction of The CEG

We now describe the sensor-based construction of the CEG during boundary traversing. First we discuss the detection of CEG nodes. Recall that the CEG nodes represent segments of convex obstacle edges which lie in $Co(B)$. During boundary traversing, the robot implicitly expands a connected surface patch $O$, which is the portion of $B$’s surface which has been seen so far. Obstacle edges which lie on $O$ can be readily detected using the range data, since they lie on a single connected surface. Next we describe how to detect convex obstacle edges which intersect $Co(B)$ but do not lie on $O$. These obstacle edges belong to $B$ or to other obstacles. We term segments of convex obstacle edges as *occluding segments* when they cause partial occlusion of $O$ from the current robot location. By definition of the boundary traversing mode, the robot is located at each step within $Co(B)$, hence the occluding segments lie in $Co(B)$. It can be verified that the union of these occluding
segments suffices for exploration of the entire surface of $B$. Hence visible convex obstacle edges are added as nodes to the CEG if they belong to $B$ or cause partial occlusion of $B$. (These two conditions are not mutually exclusive.) Note that there may be convex obstacle edges in $Co(B)$ which do not belong to $B$ and never cause partial occlusion of $B$, as shown in the 2D example of Figure 12. These edges are not included in the CEG and are never used by the algorithm.

Last we discuss the timing of the construction process within the boundary traversing motion. In the final stage of each step, the robot continuously traces a portion of a convex obstacle edge. The sensor data which is gathered during this tracing is used to update the CEG nodes. (Practically, it suffices to record sensor data only at discrete points where visual events such as the projection of a vertex on an edge take place.) After updating the CEG nodes, CEG edges are added before the robot resumes its motion.

4 Algorithm Analysis

The convergence of the 3D Bug algorithm is based on the following ideas. During motion towards the target, the distance of the robot from the target, $d(x, T)$, decreases monotonically between successive steps. Moreover, the path length of each motion-towards-the-target segment is finite. During boundary traversing, the robot either senses the entire surface of the followed obstacle, or leaves its boundary before completing the exploration. We prove that the exploration strategy always generates a finite path, thus the path length of each boundary-traversing segment is finite. The robot switches to boundary-traversing only at points which are uniquely associated with local-minima of the distance function $d(w, T)$. Since $d(w, T)$ has finitely many local-minima in any bounded polyhedral environment, there is a finite number of motion segments. As each segment is of finite length, the algorithm terminates after a finite-length path. If the target is reachable, convergence to the target is guaranteed by the leaving condition. This condition ensures that the robot always terminates its boundary traversing mode and resumes its motion towards the target. By construction, the last motion-towards-the-target segment has no obstacle blocking the robot path, and this segment leads the robot directly to $T$.

In the following, we consider a point robot in a three-dimensional space populated by a finite number of stationary polyhedral obstacles. Next we introduce some terminology. The freespace, denoted $\mathcal{F}$, is the complement of the obstacles’ interiors. The function $d(w, T): \mathcal{F} \rightarrow \mathbb{R}$ measures the Euclidean distance from a point $w$ in the freespace to the target $T$. A point where the robot switches from motion towards the target to boundary
traversing is termed a switch point \( P_i \). Each switch point has a corresponding local-minimum point \( M_i \), which is the local minimum of the function \( d(w, T) \) which triggered the switch. We term a point where the leaving condition holds and the transition phase is initiated as a leave point \( L_i \). Last, a transition point \( Z_i \) is a point where the transition phase terminates and the motion towards the target resumes.

The following two lemmas assert that each segment of the two motion modes has a finite length.

**Lemma 4.1** The path length of every motion towards the target segment is finite.

**Proof:** We show below that every motion towards the target segment consists of a finite number of focus points. First we show that the distance to the target of successive focus points \( F_i \) monotonically decreases, \( d(F_{i+1}, T) < d(F_i, T) \). Let \( D \) denote the ball of radius \( d(F_i, T) \) centered at \( T \). Let the feasible subsegment of each convex obstacle edge denote the subsegment which is included in \( D \). We term an endpoint \( e \) of a feasible subsegment as equi-distance endpoint if \( d(e, T) = d(F_i, T) \).

Consider the case in which the robot is located at a focus point \( F_i \), and it chooses the next focus point \( F_{i+1} \). Recall that the initial location of \( F_{i+1} \) is first chosen on some convex obstacle edge, and then its location may be modified along this edge. The initial location of \( F_{i+1} \) lies in the ball \( D \), for the following reasons. By construction, \( F_{i+1} \) either lies on the feasible sub-contour, on a convex obstacle edge of the blocking contour, or on a convex edge of an occluding obstacle. When \( F_{i+1} \) is chosen on the feasible sub-contour, \( F_{i+1} \in D \) holds since the feasible sub-contour is defined as the intersection of the blocking contour with \( D \). Let \( G \) be the point on the feasible sub-contour through which the locally shortest path passes. When \( F_{i+1} \) is chosen on an occluding obstacle, at the point where the line segment \([F_i, G]\) crosses the occluding edge, \( F_{i+1} \in D \) holds since the entire line segment \([F_i, G]\) is included in \( D \). Thus we have that \( F_{i+1} \in D \). Consider now the obstacle edge \( E \) which contains \( F_{i+1} \). (Note that \( E \) does not contain \( F_i \).) If there exists an equi-distance endpoint \( e \) on the edge \( E \) such that \( d(e, F_{i+1}) < \delta \), then \( F_{i+1} \) is shifted away from \( e \). By construction, we shift \( F_{i+1} \) along the obstacle edge \( E \) while decreasing \( d(F_{i+1}, T) \) until either \( d(e, F_{i+1}) > \delta \) holds, or \( F_{i+1} \) reaches \( p_E \), which has minimal distance to \( T \) along \( E \). It follows that the final location of \( F_{i+1} \) satisfies \( d(F_{i+1}, T) < d(F_i, T) \).

Next we show that only a finite number of focus points are defined on each convex obstacle edge. When a focus point \( F_i \) is located on an edge \( E \), it constitutes an equi-distance endpoint on the feasible subsegment of \( E \). The next focus point \( F_j \) chosen on \( E \) must lie in the feasible subsegment of \( E \) determined by \( F_i \). By construction, \( F_j \) either
has the minimal distance to \( T \) on the edge \( E \), or it has a distance larger than \( \delta \) from \( F_i \). (When \( \delta \) is set according to the local obstacles configuration, as explained in section 3.2, the minimal value \( \delta_0 \) should be considered here.) Hence the distance between focus points on any particular obstacle edge is larger than \( \delta \), except for one focus point which is at the point \( p_E \) closest to \( T \). It follows that the number of focus points on each obstacle edge is finite. Thus there are finitely many focus points during each motion-towards-the-target segment. Since \( F_{i+1} \) is always directly visible from \( F_i \), the robot moves between successive focus points along a straight line of finite length. Hence the path length of each motion-towards-the-target segment is finite.

**Lemma 4.2** The path length of every boundary traversing segment is finite.

**Proof:** During boundary traversing the robot incrementally constructs the CEG of the followed obstacle. We show below that the surface exploration is always completed after a finite path. In particular, if the boundary traversing is terminated earlier since a leave point is found, the path length until termination is finite. After the entire surface has been explored, the robot moves to the closest point to the target on the surface. Since this final motion is planned towards a previously seen point, its path length is also finite.

Next we show that boundary traversing consists of a finite number of steps, and that the path length of each step is finite. Recall that the entire surface is explored after the robot traced all the convex obstacle edges which are represented as nodes in the CEG. By construction, virtual CEG edges that reach the target emerge from convex obstacle edges which were not entirely traced. Moreover, each virtual edge endpoint is located next to a non-traced portion of an obstacle edge (see section 3.4.2). Hence the focus point in each step is chosen next to a non-traced portion of an obstacle edge. After reaching the focus point, the robot traces a fixed-length portion of the edge which was not traced before. Thus all the convex obstacle edges are traced after a finite number of steps. Next we show that the path length of each step is finite. The focus point \( F \) in each step lies on the CEG. Since the CEG is connected, the CEG-path towards \( F \) has a finite length. Moreover, during boundary traversing the robot performs local shortcuts relative to a predefined CEG-path. It follows that the length of each boundary traversing segment is finite.

Next we wish to show that there are finitely many segments of each motion mode. We begin with a lemma that associates with every switch point \( P_i \), a local minimum of \( d(w, T) \).

**Lemma 4.3** Every switch point \( P_i \), where the robot switches from motion towards the target to boundary traversing, has a corresponding **unique** local-minimum point \( M_i \), which is a local minimum of \( d(w, T) \), such that \( d(M_i, T) \leq d(P_i, T) \).
Figure 13: When the feasible sub-contour becomes empty, at switch point $P_i$, there exists a local minimum $M_i$ of $d(w,T)$ on the blocking surface.

**Proof:** When the robot switches to boundary traversing, there must be a blocking obstacle between the robot location at $P_i$ and $T$. Otherwise the robot can reach $T$ directly from $P_i$, and no switching to boundary traversing occurs. Let $v_{cross}$ denote the point where the line segment $[P_i,T]$ crosses the blocking surface (Figure 13). By construction $d(v_{cross},T) \leq d(P_i,T)$. The motion towards the target is terminated because the feasible sub-contour has become empty. Thus $d(P_i,T) < d(y,T)$ holds for every point $y$ on the blocking contour. It follows that $d(v_{cross},T) < d(y,T)$ for every point $y$. But $d(w,T)$ is a continuous function and the blocking surface is a connected compact set (i.e. closed and bounded), whose boundary is precisely the blocking contour. Since every continuous function attains a minimum on a compact set, there exists a local minimum of $d(w,T)$ on the blocking surface. Furthermore, the local-minimum point, $M_i$, satisfies $d(M_i,T) \leq d(v_{cross},T) \leq d(P_i,T)$. Finally, if there are several local minima, the one which is closest to $T$ is chosen, so that a unique local-minimum point is associated with $P_i$. \hfill \Box

The following lemma shows that the distance to the target monotonically decreases between successive local-minimum points.

**Lemma 4.4** The distance to the target decreases between successive local-minimum points i.e., $d(M_{i+1},T) < d(M_i,T)$.

**Proof:** According to the previous lemma, each switch point $P_i$ is associated with a unique local-minimum point $M_i$. While moving from $P_i$ to $P_{i+1}$, the robot traverses an obstacle boundary, leaves it, performs a transition phase followed by motion towards the target, and switches again to boundary traversing. During the initial boundary traversing, $d_{min}(T) \leq d(M_i,T)$ holds for the following reason. The local-minimum point $M_i$ is
visible from $P_i$ since it lies on the blocking surface, and by construction all the blocking surface is visible from $P_i$. Boundary traversing is terminated when the leaving condition, $d(v_{\text{leave}}, T) < d_{\text{min}}(T)$, holds true. The robot then performs the transition phase, and moves towards $v_{\text{leave}}$ until it reaches a point $Z_i$, which satisfies $d(Z_i, T) < d_{\text{min}}(T)$. From $Z_i$, the motion towards the target is resumed. Since the distance $d(x, T)$ decreases during motion towards the target, $d(P_{i+1}, T) \leq d(Z_i, T)$. Based on Lemma 4.3, $d(M_{i+1}, T) \leq d(P_{i+1}, T)$. Therefore $d(M_{i+1}, T) < d(M_i, T)$ holds.

The next lemma shows that if the target is reachable, the leaving condition always holds after a finite-length path.

**Lemma 4.5** If the target is reachable from a switch point $P_i$, the leaving condition will cause the robot to leave the followed obstacle after a finite-length path.

**Proof:** Since the exploration strategy generates a finite-length path (Lemma 4.2), it suffices to show that the leaving condition will be satisfied in the final target-reachability test, performed at the end of each surface exploration. The final test is performed from a point $C$ which is the closest to $T$ on the surface of the followed obstacle. At this point $d_{\text{min}}(T) = d(C, T)$. Since $T$ is reachable, it must be possible to leave the obstacle at $C$ and move towards $T$ for some, possibly short, line segment. If $T$ is visible from $C$ then $v_{\text{leave}} = T$, otherwise $v_{\text{leave}}$ is the point where the line segment $[C, T]$ crosses a different obstacle. Since $d(x, T)$ monotonically decreases along the line segment $[C, v_{\text{leave}}]$, $d(v_{\text{leave}}, T) < d_{\text{min}}(T)$ holds in both cases and the robot leaves the boundary.

The following theorem asserts that 3DBug always terminates.

**Theorem 2** Algorithm 3DBug always terminates after following a path of finite length.

**Proof:** The robot switches to boundary-traversing mode only at points which are associated with unique local-minima of $d(w, T)$ (Lemma 4.3). The distance to the target decreases between successive local-minimum points (Lemma 4.4). Thus each local-minimum point of $d(w, T)$ is associated with at most one switching to boundary traversing. There is a finite number of local minima of $d(w, T)$ in any bounded space with polyhedral obstacles. Hence the path consists of finitely many boundary traversing segments, which are interleaved by motion-towards-the-target segments and transition phases. Lemmas 4.1 and 4.2 guarantee that the path length of each motion segment is finite. The path length of each transition phase is finite since the robot moves along a straight line towards a fixed point during this phase. Hence the total path length is finite.
The following theorem asserts that $3DBug$ is complete i.e., that it always finds the target if the target is reachable.

**Theorem 3** Algorithm $3DBug$ finds the target if it is reachable from the start point.

**Proof:** As stated in Theorem 2, there is a finite number of boundary traversing segments. If $T$ is reachable from $S$, Lemma 4.5 guarantees that every boundary traversing segment will terminate. Since every such segment is followed by a transition phase, there is a last transition phase. This transition phase either terminates at the target, or it is followed by the last motion-towards-the-target segment, which consists of a straight-line path leading directly to the target. \(\square\)

## 5 Simulation Results

In this section we present simulation results which compare the path generated by the $3DBug$ algorithm to the globally shortest path. First we describe the structures employed in the simulations. These structures are the three-dimensional range-sensor simulator and the generalized visibility graph. Then we present simulation results in four environments, and describe in detail the algorithm performance in several examples. Last, we discuss the search characteristics of $3DBug$.

The three-dimensional range-sensor simulator computes the blocking surface, which blocks the way from the current robot location $x$, to the target $T$, in environments populated by general polyhedra. The output of the simulator is a polyhedral model of the blocking surface and an explicit representation of the blocking contour, which is the piecewise-linear boundary of the blocking surface. The range-sensor simulator additionally computes the obstacle edges which partially occlude the blocking surface. The simulator is based on the solid modeling package IRIT [8], which allows boolean operations between three-dimensional bodies, and it operates in the following way. If the target $T$ is visible from $x$, there is no blocking surface. Otherwise, the simulator finds the obstacle face $P$ which is visible from $x$ and intersects the line segment $[x, T]$. This face is a single planar polygon and it is possibly partially occluded. To find which edges partially occlude $P$, the simulator generates a three-dimensional pyramid, with $P$ as its base and the robot location $x$ as its tip\(^5\). If the obstacle face $P$ is completely visible the pyramid does not intersect any obstacle. If the pyramid intersects an obstacle, the intersection volume partially occludes $P$. The simulator projects the occluding volume onto $P$, and subtracts

\(^5\)For the visibility computation, we always assume that the robot is located above obstacles' surface, rather than lying on the surface.
Figure 14: The 3D generalized visibility graph for env1, consisting of a single box-like obstacle, with resolution 0.5.

this projection from the visible portion of $P$. The obstacle face $P$ which intersects the line segment $[x,T]$ is the first obstacle face tested for partial occlusion. After computing the visible portion of $P$, termed the visible patch $V$, $V$ is expanded in the following way. Every visible obstacle face $P'$ which shares edges with $V$ is tested for partial occlusion, by repeating the pyramid intersection process for $P'$. The computation is terminated when the visible patch $V$ cannot be expanded anymore.

Since computing the globally shortest path in a polyhedral environment is NP-hard [2], we approximate it by constructing and searching a three-dimensional generalized visibility graph [15], which is defined as follows. Since the shortest path in a polyhedral environment can pass through any point on obstacle edges, the obstacle edges are broken into fixed-length segments. The endpoints of these segments constitute the nodes of the 3D visibility graph, along with the start point $S$ and the target $T$. The edges of the 3D visibility graph are all the collision-free line segments which connect pairs of nodes (Figure 14). The approximate globally shortest path is computed by searching the 3D visibility graph. It can be verified that as the separation between graph nodes decreases to zero, the shortest path on the 3D visibility graph approaches the exact globally shortest path.

We now present simulation results of the 3DBug algorithm in four simulated environments. The results of the experiments are summarized in the first line of Table 1. The algorithm was tested in 400 runs for each environment, with randomly chosen pairs of start and target points. The target $T$ was always reachable but was not visible from
Figure 15: (a) A path generated by $3D Bug$ in $env1$. (b) The 3D visibility graph with resolution $0.1$ (only edges which connect $S$ and $T$ to other nodes are presented), with the approximate shortest path overlaid as a thick curve.

the start point $S$. For comparison, we calculated the shortest path on the 3D visibility graph for each run, using a resolution of $0.1$ units (the size of $env1$ is $2.5,2.5,2.5$, see figure 15(b)). In the environment $env1$, which consists of a single box-like obstacle (Figure 15), the average length of the $3D Bug$ paths was $0.999$ relative to the visibility-graph shortest paths. Moreover, the $3D Bug$ paths were shorter than the visibility-graph paths in $93\%$ of the runs. It is interesting to note that $3D Bug$ performs even better than the approximate visibility-graph, indicating its effectiveness in finding short paths to $T$. The next environment $env2$ is more complex and consists of seven box-like obstacles (Figure 16). The average path length of the $3D Bug$ paths in $env2$ was $1.02$. In both $env1$ and $env2$ the algorithm used the motion-towards-the-target mode along the entire path in over $99\%$ of the runs. (The boundary-traversing motion mode was activated only in two runs in $env2$.)

The environment $env3$ consists of a single concave obstacle, which resembles a room with a door and a window (Figures 17,18). The average path length in this environment was $1.06$ (relative to the 3D visibility graph shortest path), and the boundary-traversing mode was activated in $65\%$ of the runs. It is interesting to note that there was a significant difference between moving out of the room and moving into the room. The average path length while moving out of the room was $1.01$, while the average path of moving into the room was $1.13$. We believe that the reason for this phenomenon is as follows. Moving out from the room is easier since the exits from the room, i.e. the door and the window, are
Table 1: Simulation results of the 3DBug algorithm are shown in the first line. In the modified version of motion-towards-the-target, the robot moves towards the closest point to the target on the blocking contour, rather than moving along the locally shortest path.

<table>
<thead>
<tr>
<th></th>
<th>env1</th>
<th>env2</th>
<th>env3</th>
<th>env4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DBug</td>
<td>0.999</td>
<td>1.02</td>
<td>1.06</td>
<td>1.03</td>
</tr>
<tr>
<td>modified 3DBug</td>
<td>1.10</td>
<td>1.12</td>
<td>1.26</td>
<td>1.22</td>
</tr>
</tbody>
</table>

Figure 16: The env2 scenario. (a) The visible surfaces as they are seen from the start point S. The locally shortest path leads to $F_1$ since the blocking obstacle $O_1$ is only partially visible. (b) The path generated by 3DBug, compared to the globally optimal path.
always directly visible to the robot from its start location (Figure 17). In contrast, the robot may not see the entries to the room from its start location outside the room, and thus it has to first search for these entries (Figure 18). The last environment env4 consists of two room-like obstacles, separated by a wall (Figure 19). The start and target points were always placed on different sides of the separating wall. The performance in env4 is better than in env3 since the entries to each room are visible as the robot approach it from the other room.

In the current implementation of the boundary traversing mode, the algorithm constructs a model of the followed obstacle in addition to generating the CEG. Using this additional information, the robot can detect earlier that the entire surface of the followed obstacle has been explored, as we explain in a later example. The obstacle model consists of the union of all the obstacle surfaces which have become visible during boundary traversing. The robot concludes that the entire obstacle surface has been explored when every obstacle edge in the model have been seen from two sides, i.e. from the two polygonal faces which share this edge. Using this method, the tracing factor \( \lambda \) (i.e. the traced portion of a sub-edge) is set to zero as long as there are unvisited convex obstacle edges. If the exploration is not completed after visiting all the convex edges (in complex scenarios), the tracing factor is set to \( \lambda > 0 \), and the robot starts to trace convex obstacle edges.

The simulation results show that the locally shortest path resembles the globally optimal one in simple scenarios consisting of disjoint convex obstacles. Moreover, the algorithm generates reasonably short paths even in complex scenarios. To evaluate the contribution of the locally shortest path to the overall performance, we tested a modified version of the algorithm. In the modified version, the focus point in each step of motion-towards-the-target is chosen as the closest point to the target on the blocking contour. The results of the modified version are presented in the second line of Table 1. This modification increased the average path length relative to the original 3DBug performance, demonstrating that the locally shortest path provides a better local criterion for planning.

Next we discuss some of the reasons for the differences between 3DBug’s paths and the globally optimal ones. The locally shortest path may differ from the globally optimal one due to the limited nature of local information, as demonstrated in Figure 18. From S, the robot moves towards the “roof” of the room, since the roof is not visible from S and thus considered non-existent. After observing the roof from \( F_i \), the robot moves along the shortest path from \( F_i \) to \( T \). Moreover, the reactiveness of the motion-towards-the-target mode\(^6\) may be a disadvantage in complex scenarios, as local optimality can drive the robot

\(^6\)A reactive planner considers only the currently visible obstacles for planning the next step.
Figure 17: 3DBug in the room-like environment env3, as the robot moves out from the room. The globally optimal path is almost identical to 3DBug’s path. (a) The entire path. (b) The blocking contour computed from S, shown with a bold line. (c) The blocking contour from F₁. (d) The blocking contour from F₂.
Figure 18: $3DBug$ in the room-like environment $env3$, as the robot moves into the room. (a) The entire path. (b) The blocking contour computed from $S$, shown in bold line. (c) The blocking contour from $F_1$. (d) The blocking contour from $F_2$. The target is visible from $F_3$. 
Figure 19: 3DBug in env4. (a,b) The robot leaves house1 from the window, and enters house2 from the door. The globally optimal path is almost identical to 3DBug’s path. (c) The blocking contour from S. (d) The blocking contour from F1 (located at the internal window frame). (e) From F2 (located at the external window frame of house1), the external wall blocks the direct way to T. (f) The blocking contour from F3. The target is visible from F4.
Figure 20: (a) The robot performs a zig-zag motion during motion-towards-the-target, since its reactive motion mode does not consider obstacle surfaces which have been observed in previous steps. (b) To conclude target unreachability, the robot must explore all the six faces of the blocking obstacle.

towards currently non-visible surfaces which were visible before. A conceptual example demonstrating this phenomenon is presented in Figure 20(a). In this example the target is located inside a closed box and is therefore unreachable. From S, the rear obstacle surface is not visible, and the locally shortest path leads to X₁. From X₁ the front obstacle surface is not visible, and the locally shortest path traverses the upper obstacle surface to X₂. This pattern of motion repeats until the motion-towards-the-target mode is terminated at the point P, where the feasible sub-contour becomes empty. Note that the parameter δ, which guarantees a minimal decrease of d(x, T) during this mode, affects the number of motion-towards-the-target steps in this scenario. Increasing δ decreases the number of these steps, since the feasible sub-contour becomes empty faster. However, too large a δ, while never affecting the algorithm completeness, can lead to longer paths.

Partial occlusion is another major reason for differences between 3DBug’s paths and the globally optimal ones, as demonstrated in the example of Figure 16. Recall that the locally shortest path is based on the currently observable obstacles. Thus occluded portions of obstacles are considered as non-existent. Using the motion-towards-the-target mode, the robot moves from S to the focus point F₁, which lies on an occluding edge (Figure 16(a)). The robot chooses this path from S since it does not see the entire blocking obstacle, which is denoted O₁. When the robot reaches F₁ it observes a new portion of O₁.
The robot then computes the locally shortest path from $F_i$ to $T$ and moves along this path until reaching the target. The resulting path is distinct from the globally optimal one, which is shown in Figure 16(b). The last reason for differences between $3D Bug$'s paths and the globally optimal ones is the incorporation of the global convergence requirement during motion-towards-the-target. Restricting the computed shortest path to the feasible sub-contour may prevent the robot from moving along the precise locally optimal path, which may pass through any point on the blocking contour. Thus there are several reasons which may cause the paths of $3D Bug$ to differ from the globally optimal ones.

Now we present an example in which the robot concludes target unreachability. The target is located inside the box-like obstacle (Figure 20(b)). To conclude unreachability, the robot must explore the entire surface of the obstacle. The robot switches to boundary-traversing mode at the point $P$. It then moves to unvisited convex edges, until all the six faces of the obstacle have become visible to the robot at the point $Y$. The robot then moves to perform the final target-reachability test, and concludes that the target is unreachable.

Last we discuss the search characteristics of the $3D Bug$ algorithm. The search space of $3D Bug$ is defined as follows. Given start and target points, we consider every possible location of the robot in the workspace $x$ as a node $v$ in the search space, and connect each node $v$ to the possible locations for the next step. For comparison, we consider the classical $A^*$ algorithm which uses the 3D visibility graph as the underlying search space.\(^7\) We implemented the $A^*$ algorithm and used it for computing the shortest path from $S$ to $T$. The main advantage of $3D Bug$ is that its search space is sparser than the visibility graph, since the candidate locations for the next step are limited to a single obstacle. In particular, the next step is limited to the blocking contour during motion towards the target, and to the followed obstacle surface during boundary traversing. (The next focus point may also be located on an occluding edge, which partially occlude the blocking obstacle or the followed obstacle.) In contrast, each node $v$ in the visibility graph is connected to all the nodes which are visible from $v$, which be located on several obstacles. Consequently, the $3D Bug$ algorithm finds the target in fewer steps than $A^*$, since each node in $3D Bug$'s search space has fewer neighbors. In env3, for example, $3D Bug$ reached the target after 3.3 steps on average, while $A^*$ required 32.4 steps to reach the target. The advantage of $3D Bug$ is even more pronounced when the target is unreachable. The $3D Bug$ algorithm concludes target unreachability after exploring the entire surface of a

\(^7\)Note that $A^*$ is not appropriate for a real time sensor-based search, since the location of the most promising node, which corresponds to the current robot location, “jumps” discontinuously during the search process.
single obstacle in which the target is trapped (Figure 20(b)), while \( A^* \) must expand all the nodes in its search space to conclude unreachability. Another advantage of 3DBug is that it uses a compact data structure, since it constructs the blocking surface and its associated information in every step, based directly on the obstacles description. In contrast, a data structure which represents the entire environment may be very large. For example, the 3D visibility graph of env2 with resolution 0.1 consists of 620 nodes and 118912 edges, while 3DBug's data structure consists on the average of 7 nodes and 9 edges.

6 Concluding Discussion

We presented new results in sensor-based surface exploration and locally shortest path calculation in three-dimensional polyhedral environments. Then we incorporated these results into a new globally convergent algorithm, termed 3DBug, which navigates a point robot equipped with position and range sensors. Considering surface exploration, we showed that the entire surface of a polyhedral obstacle is visible from a finite number of view points, which are located on obstacle convex edges within the convex hull of the obstacle. Then we showed that the vertices of the shortest path in 3D lie only on convex obstacle edges. Based on these results, we described a novel data structure, termed the Convex Edges Graph or CEG, which consists of convex obstacle edges and supports both surface exploration and calculation of shortest paths. The CEG is a compact and easy to maintain data structure, relative to other data structures which were proposed for sensor based surface exploration [11, 20]. Last we introduced the notion of a locally shortest path in three dimensions, and investigated its properties. We showed that the locally shortest path must pass through the blocking contour, and used this property for a novel method for computing the approximate local shortest path, a method which is linear in the number of edges in the blocking contour.

These results were incorporated into the 3DBug algorithm. This algorithm falls within the general framework of the Bug paradigm since it strives to process the sensory data in the most reactive way possible, without sacrificing the global convergence guarantee. Moreover, 3DBug is the first Bug-type algorithm which uses three-dimensional range data and plans three-dimensional motion throughout the navigation process. During motion towards the target, the robot follows the locally shortest path in a purely reactive fashion. During traversal of obstacle surface, the robot incrementally constructs the CEG of the obstacle being followed, while performing local shortcuts based on the local range data. Simulation results show that the 3DBug algorithm generates paths which resemble the
globally shortest paths in simple scenarios of disjoint convex obstacles. Moreover, the algorithm generates reasonably short paths even in concave, room-like environments.

Let us mention several potential uses for the new algorithm. First, \( 3DBug \) can be used to navigate free-flying robots in either real tasks such as surveillance or simulated scenarios such as virtual reality games. Second, \( 3DBug \) provides insight for sensor-based navigation in three-dimensions, and constitutes a framework for motion planning in various three-dimensional configuration spaces. In particular, We intend to extend \( 3DBug \) for three-degrees-of-freedom mobile robots. In this case, well known techniques (e.g. [13] pp.126) can be employed to transform the two dimensional local range into the three dimensional configuration space \([x, y, \theta]\). \( 3DBug \) would then be applied to the \( c \)-obstacles in three dimensions.

The simulations reveal the advantages of \( 3DBug \) as a search algorithm in completely known three-dimensional configuration spaces. The main advantage of \( 3DBug \) is that its search space is sparser than typical graph structures (e.g. the three-dimensional visibility graph), since it takes into consideration the geometric characteristics of the locally shortest path. In particular, the possible locations of the next step from every node \( v \) in the search space are restricted to a single obstacle in \( 3DBug \), while all the obstacles which are visible from \( v \) are considered in the visibility graph. Consequentially, \( 3DBug \) finds the target in fewer steps. Our simulations indicate a significant improvement in the number of search steps, by a factor of 10 relative to the \( A^* \) algorithm, which was tested on the 3D visibility graph. The advantage of \( 3DBug \) is even more pronounced when the target is unreachable, since \( 3DBug \) concludes target unreachability after exploring the surface of a single obstacle in which the target is trapped. In contrast, \( A^* \) must expand all the nodes in its search space to conclude unreachability. \( 3DBug \) is very efficient in terms of memory requirements relative to \( A^* \), which uses a global model of the entire environment, since \( 3DBug \) uses a limited amount of global information. Hence \( 3DBug \) has several advantages as a search algorithm in completely known environments.
References


