3DBug: A Three-Dimensional Range-Sensor Based Globally Convergent Navigation Algorithm

Ishay Kamon and Ehud Rivlin

Dept. of Computer Science
Technion
Haifa 32000, Israel

Elon Rimon

Dept. of Mechanical Engineering
Technion
Haifa 32000,Israel

Abstract

3DBug *is a new range-sensor based algorithm for navigating a point robot in an unknown three-dimensional polyhedral environment. When moving towards the target, the robot moves in a purely reactive fashion along the locally optimal path, such that its distance to the target decreases. We develop efficient methods for computing the locally shortest path for this motion mode. When circumnavigating an obstacle, the robot uses the range data to incrementally construct the obstacle shape. The robot attempts to reach a suitable leave point along the shortest possible path, while simultaneously expanding its knowledge of the obstacle. 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, and determines that the target is unreachable. Thus* 3DBug *provides a new and effective sensor-based navigation algorithm for three-dimensions. We analyze the convergence of* 3DBug*, and present preliminary simulation results, showing that* 3DBug *produces paths that in simple environments resemble the globally shortest path.*

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 planning 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 [2, 3, 13, 15]). This approach guarantees that either the target is reached or the robot concludes that the goal is unreachable. However, the construction and maintenance of a global model based on sensory information imposes a heavy computational burden on the robot. In contrast, local path-planners use the local sensory information in a purely reactive fashion. These planners typically use navigation vector-fields which directly map the sensor readings to actions (see for example [5, 12]). 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 [8], and subsequently studied by [7, 9, 14]. The *Bug* approach minimizes the computational burden on the robot but still guarantees global convergence to the target. The robot initially moves towards the target. When it hits an obstacle, it switches to boundary following behavior. 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 for two-dimensional configuration spaces. Extending these algorithms to three-dimensions is difficult for the following two reasons. First, the obstacle boundaries are surfaces, while the robot's path is a one-dimensional curve. Thus it is not possible to reduce the test of target unreachability to loop detection. Second, a point robot moving along an obstacle boundary has an infinite number of possible directions for passing around the obstacle, in contrast to only two possibilities in the planar case. Recently, Kutulakos and coworkers studied the problem of three-dimensional sensor-based navigation [6]. They concluded that the robot must use a range sensor and maintain a model of the sensed surface to be able to conclude target unreachability. They also suggested 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 was found and the entire surface was explored, the algorithm concludes that the target is unreachable.

In previous work [4], 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 nodes. But in three-dimensional polyhedral environments it is known that 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 [1]. The shortest path can be computed in an exponential time $2^{n^{O(1)}}$, where n is the number of obstacle edges [11]. However, an ϵ-approximate shortest path can be found in a polynomial time of $O(\frac{n^4}{\epsilon^2})$, by selectively breaking obstacle edges into short segments and searching the graph with these edges as nodes [10]. We will use the ϵ-graph to efficiently search for the locally shortest path.

The new *3DBug* algorithm uses position and range sensors to navigate in a three-dimensional polyhedral environment. It falls within the general framework of the *Bug* paradigm, since it strives to process the sensory data in the most local way possible, without sacrificing the global convergence guarantee. While moving between obstacles towards the target, the robot moves in a purely reactive fashion along the locally optimal path, in a way which decreases its distance to the target. When the robot circumnavigates an obstacle, it attempts to reach a suitable leave point along the shortest possible path, while simultaneously expanding its knowledge of the obstacle 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, 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. [6], where the motion towards the target and the convergence mechanism are restricted to a plane. Thus *3DBug*

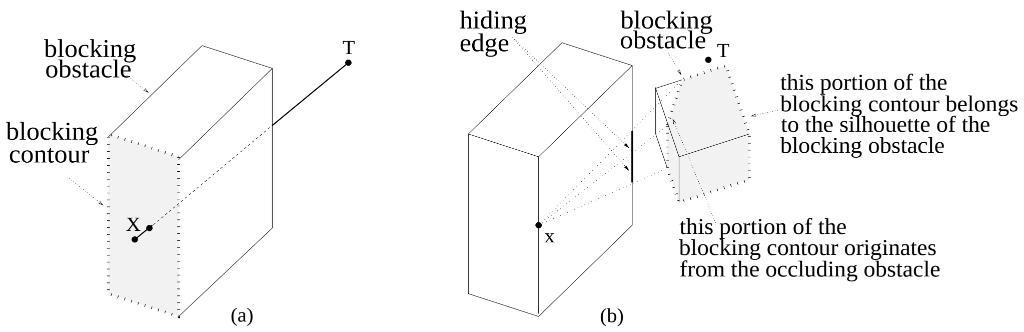

Figure 1: (a) A blocking contour consisting of silhouette edges only (b) A blocking contour consisting of both silhouette edges and edges generated from occlusion

provides a new and more effective *Bug*-type navigation algorithm for three-dimensional space.

The paper is organized as follows. In section 2 we define the locally shortest path and describe efficient methods for generating and estimating this path. In Section 3 we present the $3DBug$ algorithm. In Section 4 we discuss the convergence of $3DBug$. In Section 5 we describe preliminary 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 Locally Shortest Path in Three-Dimensions

First we define the *locally shortest path*. This is the shortest collision-free path from the robot current location x to the target T, based only on the currently observable obstacles. We assume a range sensor with infinite detection range, which provides perfect readings of the distance to the obstacles within the *visible set*. This is the three-dimensional star-shaped set centered at the robot current location x. Each sensed surface is modeled as a polyhedral two-sided thin wall (or shell) in the real world. The thin-wall model always underestimates the obstacles' size, but a more accurate model would require computationally expensive modeling techniques that we wish to avoid. 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 some visible surface of the blocking obstacle, and we refer to this surface as the *blocking surface*. The blocking surface is bounded by an occluding contour, termed the *blocking contour*, which lies on the blocking obstacle (Figure 1(a)). For polyhedral obstacles, the blocking contour consists of one or more closed piecewise linear curves. Each edge of the blocking contour either belongs to the silhouette of the blocking obstacle, or it is generated by occlusion. Each occluded edge of the blocking contour has a corresponding hiding edge on the silhouette of some other obstacle which partially occludes the blocking obstacle (Figure 1(b)).

The computational effort necessary for generating the locally shortest path is an 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. First we prove this property for two-dimensions, then extend it

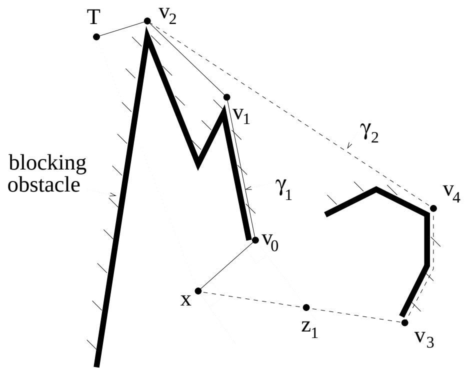

Figure 2: The shortest path from x to T, among all paths circumventing the blocking obstacle from the right, passes through the endpoint v_0 of the blocking obstacle

for three-dimensions. In two-dimensions, we assume polygonal obstacles, and model the sensed obstacles as thin one-dimensional walls.

Proposition 2.1 *Consider a planar polygonal 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 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 γ_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, γ_2, which is the shortest among all paths circumventing the blocking obstacle from the right (Figure 2). Let $Poly_1$ and $Poly_2$ be the polygons whose edges are the line segment $[x, T]$ and the polygonal paths γ_1 and γ_2, respectively.

The path γ_2 does not cross γ_1, although it may partially overlap γ_1. First, it can be verified that γ_2 may not cross γ_1 in the segment $[x, v_0]$. If γ_2 crosses γ_1 at a point c, after the segment $[x, v_0]$, there would be two possible sub-paths. One from c to T along γ_1 and the other from c to T along γ_2. However, both γ_1 and γ_2 must follow the shorter sub-path, since both paths are defined as shortest paths, and therefore cannot be shortened. Since γ_2 circumvents the blocking obstacle from the right and it does not cross γ_1, the polygon $Poly_2$ strictly includes $Poly_1$. We now show that γ_2 cannot possibly be shorter than γ_1. For every line segment $[y_i, y_{i+1}]$ of γ_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 2. Let $[z_i, z_{i+1}]$ be the line segment of γ_2 bounded by these two rays e.g., the segment $[x, z_1]$ in figure 2. Clearly $[z_i, z_{i+1}]$ is not shorter than $[y_i, y_{i+1}]$. Moreover, $Poly_1$ is convex, since γ_1 passes through convex obstacle vertices. Hence the segments $[z_i, z_{i+1}]$ are disjoint. Since $Poly_1$ is strictly included in $Poly_2$, the segments $[z_i, z_{i+1}]$ do not cover γ_2. It follows that γ_2 must be longer than γ_1. Thus the shortest path to T must pass through one of the two endpoints of the blocking obstacle. $\square$

Corollary 2.2 *Consider a polyhedral three-dimensional environment, with a blocking obstacle between the robot location x and the target T. Then the* ***shortest path*** *from x*

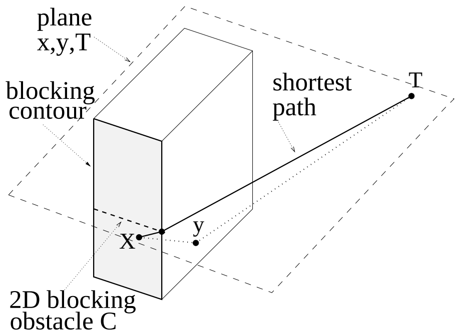

Figure 3: The shortest path from x to T passes through a point of the blocking contour

*T, **considering only the visible obstacles**, must pass through a point of the blocking contour (Figure 3).*

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 straight 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. The blocking surface intersects the plane A along a curve C, which forms a two-dimensional blocking obstacle in A (Figure 3). According to Proposition 2.1, the shortest path from x to T must pass through an endpoint of C. Since y is not an endpoint of C, there exists a shorter path from x to T in A—a contradiction. Since the endpoints of C belong to the blocking contour, the shortest path from x to T must pass through a point of the blocking contour. □

Now we discuss methods for computing the locally shortest path. Consider Papadimitriou's ϵ-graph [10]. We define the ϵ-tangent-graph, ETG, as the graph whose vertices are the ϵ-graph vertices. The edges of the ETG are the collision-free line segments connecting the vertices, such that the edges are tangent to the obstacles at their endpoints. The number of edges in the ETG is much smaller than the number of edges in the original ϵ-graph, thus searching in the ETG is more efficient. The local-ϵ-tangent-graph, LETG, is defined as the ETG which is based on the currently observable obstacles, using the thin-wall model for the obstacles, see Figure 4. Note that each thin wall has two sides, the one which is visible from the robot and the other which is hidden. It can be verified that the ϵ-approximate locally shortest path always lies on the LETG. Moreover, it can be verified that the locally shortest path passes only through vertices of the blocking surface which either lie on the blocking contour, or on the hidden side of the blocking surface. Thus it suffices to search only a small part of the LETG.

However, the construction of the LETG based on sensory information may still be computationally expensive. We now present a practical and efficient method for estimating the locally shortest path. The resulting estimate will be called the *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 $L_{con}(y) = \|x - y\| + \|y - T\|$. For each line segment l_i of the blocking contour, we compute the point v_i which minimizes $L_{con}(y)$. This computation can be done in constant time per line segment l_i. Then we construct a local graph, called the *contour graph*, consisting of edges from x to each v_i, and (optimistic) edges from each v_i to T, as shown in Figure 5.

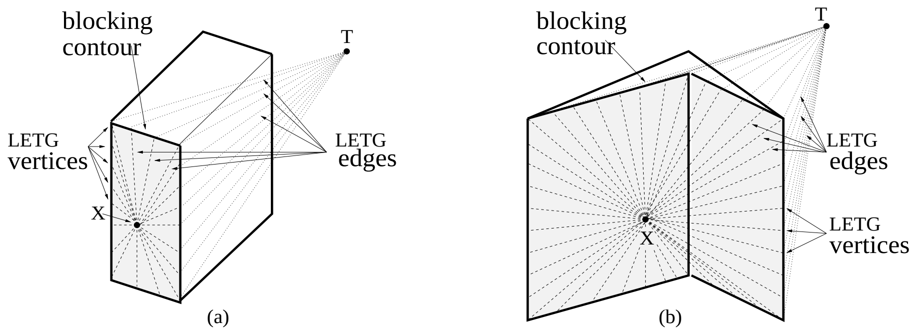

Figure 4: Examples of the LETG (a) of a convex obstacle (edges on the visible side of the blocking surface are not drown) (b) of a concave obstacle. Note that no edges lead to the concave edge of the blocking surface.

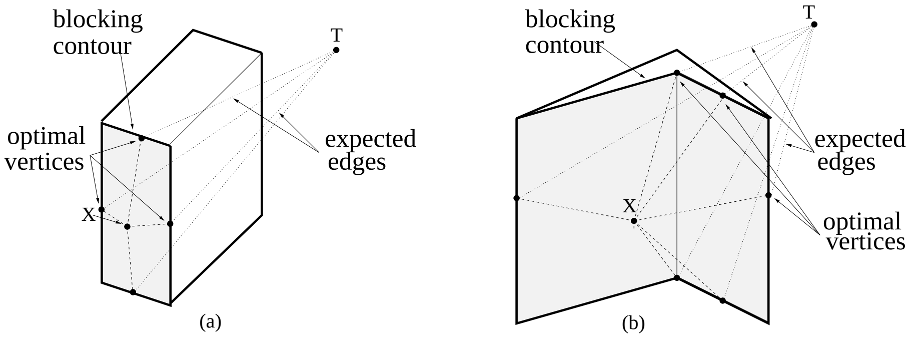

Figure 5: Examples of the contour graph, which is much smaller than the LETG graph. Contour graph of (a) a convex obstacle (b) a concave obstacle.

The contour path is the shortest path on the 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 contour path to the ϵ-approximate locally shortest path? If the blocking obstacle is convex, the contour path becomes identical to the locally shortest path, for the following reason. When the line segment $[y, T]$ intersects the blocking surface from its visible side, 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 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 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 input during the navigation consists of the robot current position x, and the distance from x to the obstacles within the visible set. First we describe the global structure of the algorithm, then discuss its detailed operation.

3.1 Algorithm Description

3DBug uses two basic motion-modes: motion towards the target and obstacle-boundary following. Further, *3DBug* operates in discrete steps as follows. At each step the robot chooses a *focus point* and moves to it, without performing any sensing or re-planning. When the robot reaches the focus point, it senses the environment and chooses the next focus point. This approach reduces the computational burden on the robot and simplifies the ensuing analysis of the algorithm.

During motion towards the target, the robot moves along the *locally optimal path* based on the currently observable obstacles. Let the function $d(w,T)$ measure the distance from a point w to T, such that the distance is measured only in the free space. The robot keeps moving towards the target until it is 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 a blocking obstacle, and the robot starts to follow the boundary of this obstacle.

During the boundary-following mode, the robot moves away from a local minimum of $d(w,T)$, but it may not leave the obstacle boundary until a suitable leave point is found. At each step the robot updates a surface model of the followed obstacle. Then it moves on the surface along the shortest path to the target according to the current surface model. At each step the robot records the minimal distance to the target, $d_{min}(T)$, observed so far on the followed obstacle. It also inspects v_{leave}, which is the closest point to the target along the visible portion of the line segment $[x,T]$. The robot leaves the obstacle boundary when $d(v_{leave},T) < d_{min}(T)$. 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_{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.

When the robot completes the exploration of the entire obstacle surface without finding a suitable leave point, it performs the following final target-reachability test. The robot moves to the closest point to the target on the surface, and checks the leaving condition from there. If the leaving condition is not satisfied, the target is unreachable. A summary of the algorithm now follows.

1. Move towards T along the *locally optimal path*, until one of the following events occurs:
   - The target is **reached**. Stop.
   - A local minimum is detected. Go to step 2.
2. Follow the obstacle's surface along the shortest path to T according to the current model, while updating the surface model and recording $d_{min}(T)$, until one of the following events occurs:
   - The target is **reached**. Stop.
   - The leaving condition holds: $\exists v_{leave}$ *s.t.* $d(v_{leave},T) < d_{min}(T)$. 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 $d(z,T) < d_{min}(T)$. Go to step 1.

3.2 Motion Towards the Target

During motion towards the target, the robot moves along the locally optimal path based on the currently sensed obstacles. Moreover, we wish to guarantee that the distance of the robot to the target decreases monotonically during this motion. The locally shortest path passes through the blocking contour. Hence, to guarantee a decreasing distance to the target, we compute the locally optimal path based only on the points y of the blocking contour satisfying $d(y,T) < d(x,T)$, where x is the robot location. This subset of the blocking contour is termed the *feasible sub-contour*. In other words, the feasible sub-contour is the intersection of the blocking contour with the ball of radius $d(x,T)$ centered at T, denoted $\mathcal{B}$. Recall that the blocking contour lies on the blocking obstacle, its edges are either silhouette edges or obstructed edges, and each obstructed edge has a corresponding hiding edge. Let the *hiding set* be the intersection of the hiding edges with $\mathcal{B}$. For a reason explained below, we further restrict the feasible sub-contour as follows. Consider those endpoints of the feasible sub-contour which lie on the boundary of $\mathcal{B}$, called the *open* endpoints. At each open endpoint, we shorten the corresponding edge of the feasible sub-contour by δ, where δ is a small positive parameter. When a hiding edge E_H is shortened, the corresponding obstructed edge E_O is also shortened in the following way. Let $E_H^{'}$ denote the shortened hiding edge. to a reduced edge $E_H^{'}$ hiding edge. Only the portion of E_O which is obstructed by $E_H^{'}$ belongs to the feasible sub-contour. The trimming excludes cases where the locally optimal path passes infinitesimally close to open endpoints. Such cases can lead to an infinite path length that we wish to avoid.

If the target is visible to the robot, the shortest path leads directly towards it. Otherwise, the algorithm constructs a local ϵ-tangent-graph (LETG) from the feasible sub-contour and the target node, and searches the LETG for the shortest path to T. In our current implementation we use the smaller contour-graph to search for the shortest path. However, recall that the resulting contour-path is only an estimate for the true locally-shortest path. Once the locally shortest path is computed, the focus point F is chosen on this path as follows. Let G denote the point on the feasible sub-contour through which the locally optimal path passes. If G lies on an edge of the blocking obstacle, F is set to G (Figure 6(a)). If G lies on an edge generated from occlusion by some other obstacle, F is chosen on the hiding obstacle, at the point where the line segment $[x,G]$ crosses the hiding edge (Figure 6(b)). The focus point is not chosen on the edge of the feasible sub-contour in this case, since the globally shortest path never passes through such edges. Since we wish to achieve local decisions that resemble the globally optimal ones as closely as possible, it makes sense to locate F on the hiding edge.

The motion towards the target terminates when the robot detects that it is trapped in the basin of attraction of a local minimum of the function $d(w,T)$. As we show in the ensuing analysis, this event occurs precisely when the feasible sub-contour becomes empty. The robot subsequently terminates its motion towards the target and switches to the boundary-following mode. Note that the switch may occur even at the start point S, as illustrated in Figure 7.

3.3 Following an Obstacle Surface

The local minimum of $d(w,T)$ which terminated the motion towards the target is necessarily visible from the robot current location. Moreover, this local minimum lies on the

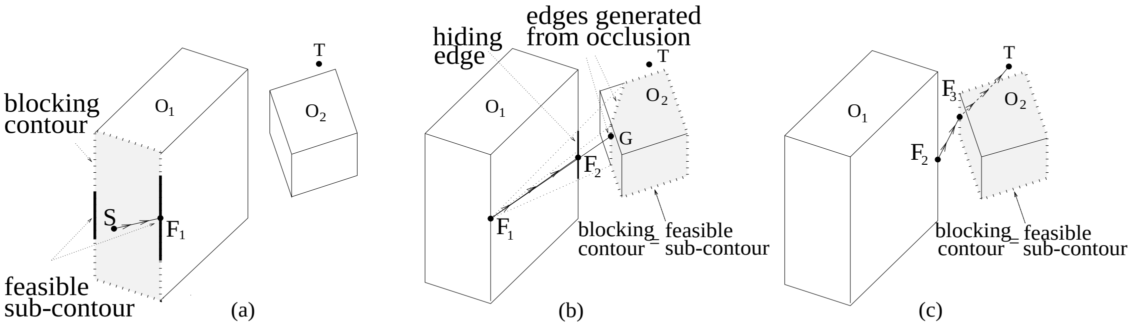

Figure 6: Examples of motion towards the target. (a) From S, the locally optimal path passes through the focus point F_1. (b) At F_1 two edges of the blocking contour result from occlusion. The point G lies on an occluded edge, hence the next focus point, F_2, is placed on the hiding edge. (c) From F_2 the locally optimal path passes through F_3, from which the target can be reached directly.

surface of the current blocking obstacle. The robot follows this surface until a leaving condition is satisfied or the entire surface is explored. At each step the robot updates the surface model of the currently followed obstacle. Then the robot calculates the shortest path to the target according to the current surface model, as described below. Given the shortest path, the robot chooses the focus point, F, at the furthest point along the shortest path which lies on the known portion of the surface. The robot then moves to F.

The calculation of the shortest path is integrated with an exploration of the surface as follows. At each step the robot constructs the ϵ-tangent-graph (ETG) of the currently known surface together with the target node. (Note that only the surface of the currently followed obstacle is considered.) We will refer to each vertex of the ETG as a single point, although the vertices of the ETG are short segments. To guarantee surface coverage, the edges emanating from the target are defined in the following modified way. We keep *every* edge from the target to a vertex v of the ETG which is tangent to the obstacle at v, even if the edge is not collision-free. The weight of each such edge is set to the distance $\|v - T\|$. This weight may be updated by the algorithm as we now explain.

At each step the robot tests the leaving condition as follows. After updating the surface model, the robot records the minimal distance to the target, $d_{min}(T)$, observed so far on the surface. Then it inspects v_{leave}, the closest point to the target along the visible portion of the segment $[x, T]$ (Figure 7). The leaving condition is satisfied when $d(v_{leave}, T) < d_{min}(T)$. If the leaving condition is not satisfied at the robot current location, and an edge to the target exists at this location, the robot updates the weight of this edge to ∞. As discussed below, this mechanism guarantees that *the robot will explore the entire obstacle surface*, since all the vertices v which are next to unexplored faces have $[v, T]$-edges finite weight.

Target unreachability is concluded only after the robot explores the entire surface of the obstacle without finding a suitable leave point. The robot then moves to the closest point to the target on the surface, and checks the leaving condition from there. If the leaving condition is not satisfied from that point, the target is unreachable. The final test is necessary since the leaving condition is tested only at a discrete set of points (the vertices of the ETG), and these points do not suffice to conclusively determine target

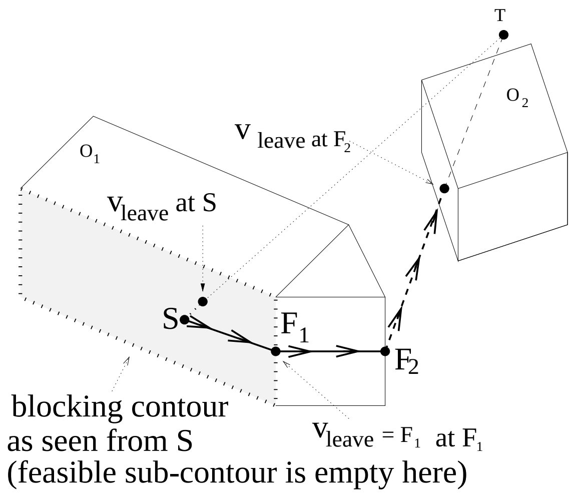

Figure 7: An example of boundary following. The robot switches to boundary following at S, since $d(S,T) < d(y,T)$ holds for every point y on the blocking contour. The locally optimal path leads to F_1, then to F_2. From F_2 the blocking obstacle is O_2 and the leaving condition holds.

unreachability.

Finally, 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_{leave} until it reaches a point z where $d(z,T) < d_{min}(T)$. As discussed below, the transition phase guarantees that the distance of the local-minima to the target decreases at successive switch points.

4 Algorithm Analysis

The convergence of $3DBug$ is based on the following ideas. During motion towards the target, the distance from the robot location x to T, $d(x,T)$, decreases monotonically between successive steps and the path length during this mode is guaranteed to be finite. The robot switches to boundary-following mode only at points which are uniquely associated with local-minima of $d(w,T)$. Since there are finitely many local-minima, there is a finite number of motion segments. In every boundary following segment, the robot either senses the entire surface of the followed obstacle or leaves its boundary before completing the exploration. We will prove that the exploration strategy generates a finite path. Hence the algorithm terminates after a finite path. If the target is reachable, convergence to the target is guaranteed by the leaving condition. This condition, when checked in the final test, ensures that the robot terminates its boundary following mode and resumes its motion towards the target. The last such motion takes the robot to T. In the following, we consider a point robot in a three-dimensional configuration space (*c-space*) populated by a finite number of polyhedral obstacles. The free c-space, denoted $\mathcal{F}$, is the complement of the obstacles' interiors. The function $d(w,T) : \mathcal{F} \rightarrow I\!R$ measures the Euclidean distance from a point w in the free space to the target T.

We define several distinguished points along the path. A switch point P_i is a point where the robot switches from motion towards the target to boundary following; each switch point has a corresponding minimum point M_i, which is a local minimum of the function $d(w, T)$. A leave point L_i is a point where the leaving condition holds and the transition phase is initiated. Last, a transition point Z_i is a point where the transition phase terminates and the motion towards the target is resumed.

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

Proof: Every step during motion towards the target has a finite length, because the robot moves in a straight line towards a visible focus point. We will show that the number of focus points is finite.

First note that the distance from focus points F_i to the target decreases monotonically, $d(F_{i+1}, T) < d(F_i, T)$. When F_{i+1} is chosen on a silhouette edge, this property holds because all the points y of the feasible sub-contour satisfy $d(y, T) < d(F_i, T)$. When F_{i+1} is chosen on a hiding edge, as the crossing point of the line $[F_i, G]$ with the occluding obstacle, this property holds because G lies on the feasible sub-contour, and thus the entire line $[F_i, G]$ is included in $\mathcal{B}$, where $\mathcal{B}$ denote the ball of radius $d(F_i, T)$ centered at T.

Let the *feasible part* of an obstacle edge $[u, v]$ denote the segment which lies on $[u, v]$, is included in the ball $\mathcal{B}$ and undergoes trimming of open endpoints. The trimming process guarantees that the length of the feasible part is reduced by at least δ whenever a focus point F_i is placed on the edge $[u, v]$. The feasible part does not increase because the ball $\mathcal{B}$ shrinks between successive steps. Since focus points are placed in the feasible part of edges, the number of focus points which may be defined on each obstacle edge is finite. It follows that there is a finite number of focus points, and therefore the path length of every motion towards the target segment is finite. □

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

Proof: Every boundary following segment consists of two phases. During the exploration phase, the robot incrementally constructs the surface model of the followed obstacle. During the final phase, after the entire surface was sensed, the robot moves to the closest point to the target on the surface. It is sufficient to prove that the exploration phase is completed after a finite path. Note that the exploration phase may be terminated, at every step, by the leaving condition.

The entire surface is sensed after a finite number of updates, because at least one face becomes visible in every update, and each polyhedral obstacle has a finite number of faces. Let the *rim* denote the set of model edges which are next to unexplored faces. All the vertices of the ϵ-tangent-graph which lie on the rim, v, have $[v, T]$ edges which are initialized with finite weights. When the robot reaches a rim vertex, it updates the surface model.

Next we show that the path length between successive updates is finite. At each step, the robot calculates the shortest path to T on the current ETG and moves to the furthest vertex on the path, v_L. The robot reaches v_L in a finite path. Since v_L is the last vertex on the path, it has an edge $[v_L, T]$ with a finite weight. If the leaving condition holds from v_L then the boundary following is terminated. Otherwise, the weight of the edge $[v_L, T]$ is updated to ∞, so that v_L will not be used again as the last vertex on the shortest path

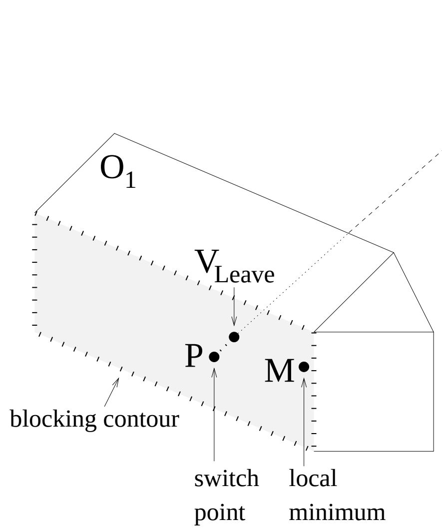

Figure 8: When the feasible sub-contour becomes empty, at switch point P, there is a local minimum M of $d(w,T)$ on the blocking surface.

to T. Since there is a finite number of vertices in the current ETG which do not lie on the rim, the robot will reach a rim vertex in a finite path. It will then update the surface model. Since the number of updates is finite, it follows that the exploration phase will be completed in a finite path. □

Lemma 4.3 *Every switch point P_i, where the robot switches from motion towards the target to boundary following, has a corresponding **unique** minimum point M_i, which is a local minimum of $d(w,T)$.*

Proof: When the robot switches to boundary following, there must be a blocking obstacle between the robot location P_i and T, otherwise the robot would reach T directly. In this situation the point v_{leave} is the crossing of the line $[P_i,T]$ with the blocking surface (Figure 8). Clearly, $d(v_{leave},T) \leq d(P_i,T)$ holds. The motion towards the target is terminated because the feasible sub-contour becomes empty. Therefore, $d(P_i,T) < d(y,T)$ holds for every point y on the blocking contour. It follows that $d(v_{leave},T) < d(y,T)$ for every point y, hence there is a local minimum of $d(w,T)$ on the blocking surface. Furthermore, the local minimum point M_i satisfies $d(M_i,T) \leq d(v_{leave},T) \leq d(P_i,T)$. By definition, all the blocking surface is visible from P_i, and in particular the local minimum is visible. If there are several local minima, the one which is closest to T is chosen. □

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

Proof: Each switch point P_i is associated with a minimum point M_i. While moving from P_i to P_{i+1}, the robot follows an obstacle boundary, leaves it, performs a transition phase followed by motion towards the target, and switches again to boundary following. During the initial boundary following, $d_{min}(T) \leq d(M_i,T)$. The leaving condition holds when v_{leave} satisfies $d(v_{leave},T) < d_{min}(T)$. In the transition phase the robot moves towards v_{leave} to Z_i, which satisfies $d(Z_i,T) < d_{min}(T)$. The motion towards the target is resumed in Z_i. 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. $\square$

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 path (Lemma 4.2), it is sufficient to prove that the leaving condition will be satisfied in the final test. The final test is performed from a point C which is closest to T on the surface of the followed obstacle. At this point $d_{min}(T)$ is updated to $d(C,T)$. Since T is reachable, it must be possible to move from C directly towards T. If T is visible from C then $v_{leave} = T$, otherwise v_{leave} lies on a different obstacle. In both cases $d(v_{leave},T) < d_{min}(T)$ holds and the robot leaves the boundary. $\square$

The following theorem asserts that $3DBug$ always terminates.

Theorem 1 *Algorithm $3DBug$ always **terminates** after following a path of finite length.*

Proof: The robot switches to boundary-following mode only at points which are uniquely associated with local-minima of $d(w,T)$ (Lemma 4.3). The distance to the target decreases between successive minima points (Lemma 4.4). Thus each local-minimum point of $d(w,T)$ is associated with at most one switching to boundary following. There is a finite number of local minima of $d(w,T)$ in any c-space with polyhedral obstacles. Hence the path consists of finitely many boundary following segments, which are interleaved between motion towards the target segments and transition phases. Lemmas 4.1 and 4.2 guarantees that the path length of each motion segment is finite. The path length of each transition phase is finite because the robot moves towards a fixed point. Hence the total path length is finite. $\square$

The following theorem asserts that $3DBug$ is complete.

Theorem 2 *Algorithm $3DBug$ **finds the target** if it is reachable from the start point.*

Proof: As stated in Theorem 1, there is a finite number of boundary following segments. If T is reachable from S, Lemma 4.5 guarantees that every boundary following segment terminates when the leaving condition holds. 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 terminates at the target. $\square$

5 Preliminary Simulation Results

We implemented the $3DBug$ algorithm, and tested it on several environments with convex obstacles. The results are promising, since the algorithm generates paths that resemble the globally shortest paths in most of the cases. We briefly discuss the results in two simulated environments. In the first environment the obstacles are relatively small (Figure 9). Consequently, the robot has a small probability to be trapped in the basin of attraction

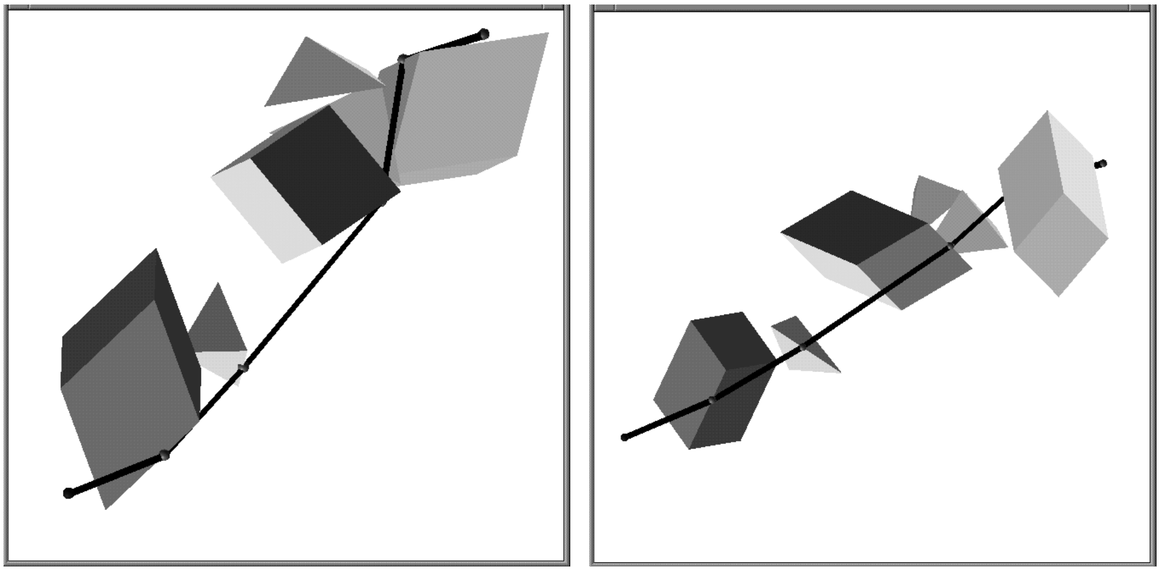

Figure 9: A simple simulation example

of local minima. In this environment, the algorithm usually uses only the motion towards the target behavior, and generates globally optimal paths. In the second environment the obstacles are larger (Figure 10). Therefore the robot has a larger probability to activate the boundary-following behavior. In this environment there are more cases where a large part of an obstacle is occluded. In these cases the locally shortest path may differ significantly from the globally shortest one (Figure 10). Thus the generated path depends more on the local arrangement of the obstacles. We consider these examples as preliminary results. A more complete implementation of the algorithm is under construction.

6 Concluding Discussion

We presented *3DBug*, a new range-sensor based globally convergent algorithm for navigating a point robot in an unknown three-dimensional polyhedral environment. *3DBug* uses 3D range data in the most reactive fashion possible and plans 3D motion throughout the navigation process. Thus *3DBug* provides a new and effective *Bug*-type navigation algorithm for three-dimensions. During motion towards the target, the robot follows the locally shortest path, and we presented an efficient method for generating it. We also presented a novel method for estimating the locally shortest path, in time which is linear in the number of line segments in the blocking contour. The preliminary simulations show promising results, as *3DBug* generates paths that often resemble the globally shortest path.

Let us mention several potential uses for the new algorithm. First, *3DBug* provides insight for sensor-based navigation in three-dimensions, and constitutes a framework for motion planning in various three-dimensional configuration spaces. *3DBug* can be used to navigate free-flying robots in either real tasks such as surveillance or simulated scenarios such as virtual reality games. We intend to adjust *3DBug* for three-degrees-of-freedom mobile robots. The algorithm can also be incorporated in motion planning for higher

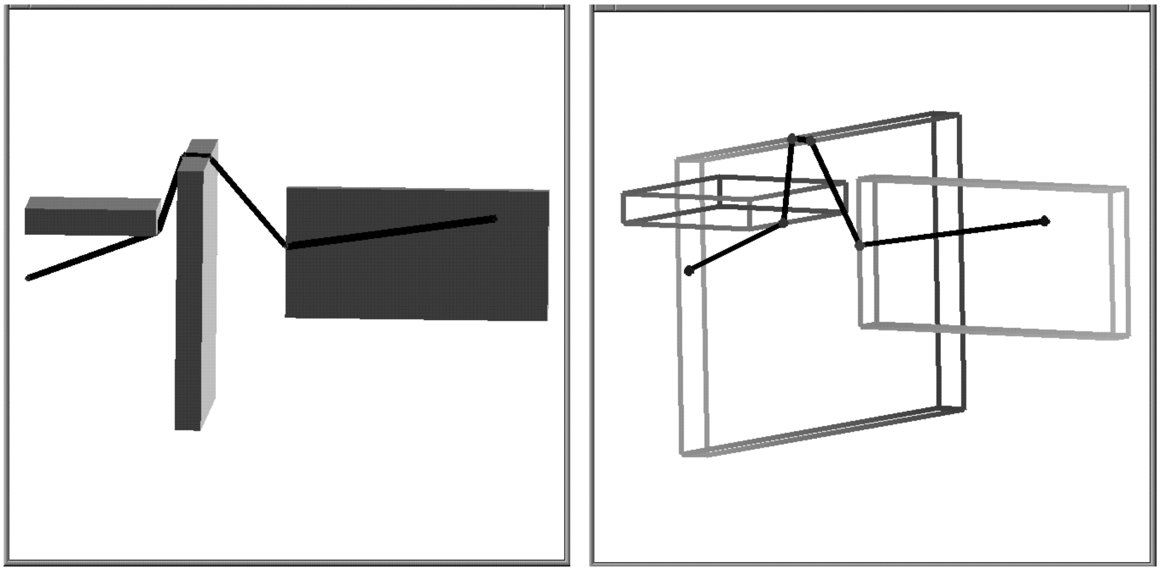

Figure 10: In this example the locally optimal path from S differs from the globally shortest one, because it is based only on the visible obstacles.

degrees of freedom robots. For example, for planning the three-dimensional trajectory of an end-effector or a sensing device mounted on a manipulator arm.

References

[1] J. F. Canny. *The complexity of robot motion planning.* MIT Press, Cambridge, MA, 1988.

[2] H. Choset and J. W. Burdick. Sensor based planning, part ii: Incremental construction of the generalized voronoi graph. In *IEEE Conference on Robotics and Automation*, Nagoya, Japan, May 1995.

[3] G. Foux, M. Heymann, and A. Bruckstein. Two dimensional robot navigation among unknown stationary polygonal obstacles. *IEEE Transactions on Robotics and Automation*, 9(1):96–102, 1993.

[4] I. Kamon, E. Rimon, and E. Rivlin. A new range-sensor based globally convergent navigation algorithm for mobile robots. In *IEEE Conference on Robotics and Automation*, pages 429–435, 1996.

[5] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *IEEE Conference on Robotics and Automation*, pages 500–505, 1985.

[6] K. N. Kutulakos, V. J. Lumelsky, and C. R. Dyer. Vision guided exploration: a step toward general motion planning in three dimensions. In *IEEE Conference on Robotics and Automation*, pages 289–296, 1993.

[7] V. J. Lumelsky. A comparative study on the path length performance of maze-searching and robot motion planning algorithms. *IEEE Transactions on on Robotics and Automation*, 7(1):57–66, 1991.

[8] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst obstacles of arbitrary shape. *Algoritmica*, 2:403–430, 1987.

[9] H. Noborio and T. Yoshioka. An on-line and deadlock-free path-planning algorithm based on world topology. In *IEEE/RSJ Conference on Intelligent Robots and Systems, IROS*, pages 1425–1430, 1993.

[10] C. H. Papadimitriou. An algorithm for shortest path motion in three dimensions. *Information processing letters*, 20:259–263, 1985.

[11] J. Reif and J. Storer. Shortest paths in euclidean space with polyhedral obstacles. Dept. of Computer Science CS-85-121, Brandeis University, 1985.

[12] D. Reznik and V. J. Lumelsky. Sensor-based motion planning in three dimensions for a highly redundant snake robot. *Advanced robotics*, 9:255–280, 1995.

[13] E. Rimon and J. F. Canny. Construction of c-space raodmaps from local sensory data. what should the sensors look for ? In *IEEE Conference on Robotics and Automation*, pages 117–123, 1994.

[14] A. Sankaranarayanan and M. Vidyasagar. Path planning for moving a point object amidst unknown obstacles in a plane: the universal lower bound on worst case path lengths and a classification of algorithms. In *IEEE Conference on Robotics and Automation*, pages 1734–1941, 1991.

[15] A. Stentz. Optimal and efficient path planning for partially known environments. In *IEEE Conference on Robotics and Automation*, pages 3310–3317, 1994.