Approximate Free Space Construction and Maximum Clearance Path Planning for a Four Degree of Freedom Robot

Chloe Arluck* Victor Milenkovic† Elisha Sacks‡

Abstract

We present an algorithm for constructing an inner approximation of the free space for a polyhedral robot with four degrees of freedom. The robot rotates about a fixed axis and translates in three dimensions with respect to a fixed polyhedral obstacle. We approximate the free space by subdividing the rotation dimension into short angle ranges, generating a three dimensional free space for each angle range, and constructing a graph for navigation in the four dimensional space. We also present an algorithm for path planning that is complete in the approximated space. The path planning algorithm produces paths that are guaranteed to be collision free and approximately maximizes obstacle clearance, ensuring safe and practical paths.

1 Introduction

We present a method of approximating the free space of a polyhedron R that rotates around a fixed axis, without loss of generality the z-axis, and translates freely relative to a polyhedron O. For example, R models a drone helicopter and O models a warehouse. Further, we present a path planning algorithm that demonstrates the benefit of free space construction for fast and efficient navigation: after a one-time computation to construct the free space, we can quickly construct paths between any two configurations or determine that none exists.

There are cases where traditional probabilistic road map planners have poor performance, particularly cases where the path must traverse through narrow passages. Additionally, PRM planners test for collision along local paths by taking discrete samples, and therefore may miss collisions when obstacles are very thin. While many sampling methods have been developed to address the narrow passage problem [1] [2] [5], and collision detection can be guaranteed using adaptive resolution [15], the challenge of choosing the appropriate variety of PRM planner for the problem adds additional tuning complexity for the user. We present a path planning algorithm that requires minimal tuning, handles narrow passages, and produces paths that are always collision free.

*Department of Computer Science, University of Miami c.arluck@cs.miami.edu

†Department of Computer Science, University of Miami vjm@cs.miami.edu

‡Department of Computer Science, Purdue University elisha.sacks@gmail.com

2 Constructing the Free Space

We approximate the four dimensional free space S by subdividing the rotation dimension into short angle ranges and generating a three dimensional free space S_i for each angle range. If n is the number of angle ranges, each S_i is an inner approximation of the free space with rotation $[\frac{2\pi i}{n}, \frac{2\pi(i+1)}{n}]$. Meaning, for all $\theta \in [\frac{2\pi i}{n}, \frac{2\pi(i+1)}{n}]$ and for all $t \in S_i$, R does not intersect O at (t, θ), where (t, θ) denotes a rotation of R by θ then a translation of R by t. To generate S_i, we construct R', a polyhedral approximation of R swept through the rotation $[0, \frac{2\pi}{n}]$ about the z axis. Then $S_i = \overline{O \oplus -R'_i}$ where R'_i is R' rotated $\frac{2\pi i}{n}$ about the z axis, $\oplus$ denotes the Minkowski sum, minus denotes the negation of each vertex, and over line denotes the complement.

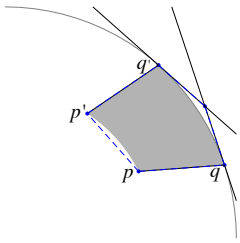
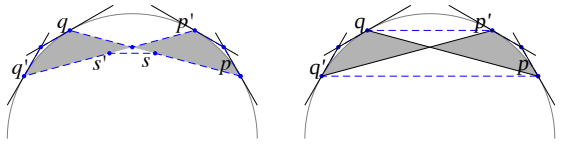


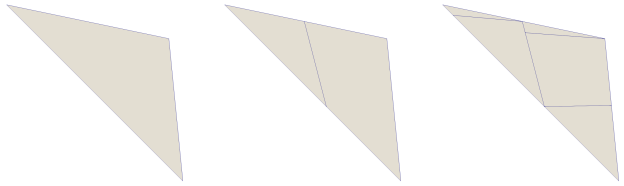
Figure 1: The containing polygon of a segment pq rotated in the plane.

To construct the outer approximation of the sweep of R, we reduce the problem to sweeping segments in the plane. Let p and q be endpoints of a segment in the plane where $|p| \leq |q|$. If p' and q' are p and q rotated θ about the origin, then an outer approximation of the sweep for segment pq is the pentagon with endpoints p, p', q, q', and the intersection of tangent lines at q and q' (Fig. 1). If the nearest point on pq to the origin is not an endpoint, we split at the nearest point and take the sweep of the two segments individually. Fig. 2 shows that not splitting at the nearest point results in a poor approximation.



(a) (b)

Figure 2: (a) The union of polygons containing the sweep of ps and qs, where s is the point on pq closest to the origin, and (b) a single polygon containing the sweep of pq.

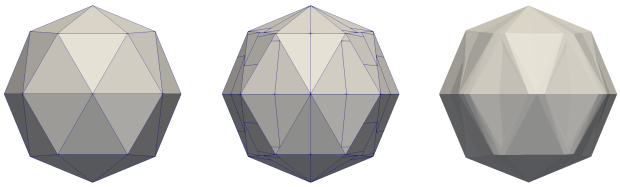


(a) (b) (c)

Figure 3: (a) An input face F and the result of splitting F by (b) its tangent plane and then by (c) xy-planes passing through each vertex.

This two dimensional method is used to construct the outer approximation of the sweep of R as follows. Let F be a face of triangulated polyhedron R. We subdivide F in order to consider it as a set of segments pq that lie on planes parallel to the xy-axis. We split F by the plane containing the nearest point of each segment (Fig. 3(b)) to avoid the poor approximation shown in Fig. 2. Next, we split by planes parallel to the xy-plane going through each vertex (Fig. 3(c)). The result is a set of trapezoids each with their top and bottom edges being segments parallel to the xy-plane.

The outer approximation of the sweep of a trapezoid is the union of the 2D sweep approximations for each z cross section. Sides of this union are ruled but not necessarily planar. We introduce cross section segments separated by at most θ in angle and $r\theta$ in z, where r is the radius of the robot, and we connect adjacent segments by their convex hull. The outer approximation of the sweep of R is its union with the outer approximation of each trapezoid sweep.

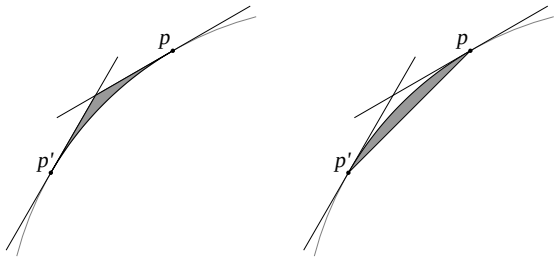


(a) (b) (c)

Figure 4: (a) A triangulated robot R, (b) its tetrahedralization and (c) its sweep polyhedron R'.

Given the three dimensional free spaces S_i, we construct a graph for navigation in the four dimensional space S. A node represents a connected component of a space S_i. If components of S_i and S_{i+1} intersect, then their corresponding nodes are neighbors. By providing the relationship between the inner approximated subspaces S_i, this graph defines an inner approximation of S. After performing the one-time computation to construct the inner approximation of S and the graph, we can query the graph to quickly construct paths between any two configurations, or determine that none exists.

3 Error Bound on the Sweep Polyhedron



(a) (b)

Figure 5: (a) A *cap* and (b) a *cup* at point p.

We wish to find an upper bound on the excess generated by this sweep approximation. For a sweep of a segment the excess is composed of *caps* and *cups*. A cap at a point p is the region between a circle of radius $|p|$ and the tangent lines at p and p', where p' is p rotated θ about the origin (Fig. 5). The area of a cap is $p^2(\tan\frac{\theta}{2} - \frac{\theta}{2})$, which has third degree Taylor series approximation $p^2\frac{\theta^3}{24}$. A cup at p is the region between segment pp' and a circle of radius $|p|$ (Fig. 5). A cup at p has area $p^2\frac{\theta-\sin\theta}{2}$ and Taylor series approximation $p^2\frac{\theta^3}{12}$.

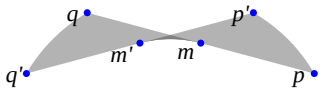


Figure 6: The pseudo area when the midpoint m is the closest point on pq to the origin. In this case equivalent to the actual sweep area.

To find an upper bound on the excess area of the sweep, we consider the worst case. For convenience, we define the *pseudo area*, a measure that is always less than or equal to the actual area swept by segment pq. If m is the midpoint of segment pq, then the pseudo area is defined to be

$$(q^2 + p^2 - 2m^2)\frac{\theta}{2} - m^2\frac{\theta^3}{24}.$$

Intuitively, the pseudo area is the sum of the areas swept by the two subsegments on either side of m without a cap at m. When m is the closest point on pq to the origin, the pseudo area is equivalent to the actual sweep area (Fig. 6) and we will show that it is always less than or equal to the actual sweep area.

If p is the point on the segment closest to the origin, then the area swept by pq is

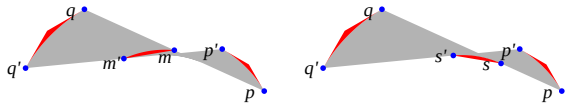
$$\int_{|p|}^{|q|} \theta r dr = (q^2 - p^2)\frac{\theta}{2}.$$

Since p is the closest point on pq to the origin, $|m| > |p|$. Hence $q^2 - p^2 > q^2 + (p^2 - m^2) - m^2$ and the pseudo area must be smaller than the actual area in this case.

If the closest point on pq to the origin is some internal point s, then the area swept by the segment is the sum of the areas swept by the two sub-segments without a cap at s, which is swept by both sub-segments.

$$(q^2 + p^2 - 2s^2)\frac{\theta}{2} - s^2\frac{\theta^3}{24}$$

Since, $|m| \geq |s|$, this area must be greater than or equal to the pseudo area.



(a) (b)

Figure 7: The (a) pseudo excess and (b) actual excess generated when approximating the sweep of segment pq, where m is the midpoint and s is the point closest to the origin.

Similar to the pseudo area, we define the *pseudo excess*, a measure that is always greater than or equal to the excess area generated by the outer approximation of sweeping segment pq. The pseudo excess is a cap at endpoints p and q and a cup at midpoint m (Fig. 7).

$$(q^2 + p^2 + 2m^2)\frac{\theta^3}{24}$$

If p is the point closest to the origin, then the excess is a cap at q and a cup at p.

$$(q^2 + 2p^2)\frac{\theta^3}{24}.$$

Since $|m| \geq |p|$, this value must be smaller than the pseudo excess. If s is the closest point on pq to the origin, then the excess is a cap at p, a cap at q, and a cup at s.

$$(q^2 + p^2 + 2s^2)\frac{\theta^3}{24}$$

Since $|m| > |s|$, this value is greater than or equal to the pseudo excess.

Now, given triangulated polyhedron R, consider an input face. The outer approximation algorithm splits by a plane parallel to the xy-axis at each vertex, so the input face is divided into two triangles sharing an edge pq with length l. We wish to find the vertex positions that result in the smallest pseudo area and the largest pseudo excess, while fixing the z-components of each vertex and the length of segment pq.

Without loss of generality we consider only one of the two triangles. Let t be the vertex opposite edge pq and consider an arbitrary segment $\hat{p}\hat{q}$ where triangle pqt intersects a plane parallel to the xy-plane. Since the z-components of p, q, and t and the length of pq are fixed, the length of $\hat{p}\hat{q}$ is also fixed. Let $\hat{m}$ be the midpoint of $\hat{p}\hat{q}$. If the distance from the z-axis of every $\hat{m}$ does not decrease, then for segment $\hat{p}\hat{q}$ the pseudo area does not increase and the pseudo excess does not decrease. We will perform a series of operations that do not decrease the distance between $\hat{m}$ and the z-axis and that fix the z-components of all vertices and the length of pq.

1. Rotate t about the z-axis until it is aligned with the midpoint of pq. Since t is moving towards the midpoint of pq, $\hat{m}$ is moving away from the z-axis.

2. Rotate p about q and until $|p| = |q|$ while rotating t about the z-axis to remain aligned with the midpoint of pq. Since p is moving away from the z-axis and and t remains aligned with the midpoint, $\hat{m}$ is moving away from the origin.

3. Scale the x and y components of t until t is distance r from the z-axis, where r is radius of the robot.

4. Translate p and q to distance r from the z-axis such that the distance l between p and q stays the same. Then $m^2 = r^2 - \frac{l^2}{4}$, the pseudo area is $\frac{l^2\theta}{4} + (\frac{l^2}{4} - r^2)\frac{\theta^3}{24}$, and the pseudo excess is $(4r^2 - \frac{l^2}{2})\frac{\theta^3}{24}$. As r increases, the pseudo area decreases and the pseudo excess increases.

The result is a scalene triangle where all vertices are at the maximum distance from the origin and t is equidistant to p and q. Changing the position of any vertices either reverses one of the above operations, exceeds the radius of the robot, or violates the original conditions that the z-components and length of pq are fixed. Hence this triangle achieves the minimal allowable pseudo area and the maximal allowable pseudo excess. Since t is equidistant to p and q, the midpoint of each segment is the nearest point, so the pseudo excess and area are equal to the true excess and area. Since the pseudo excess and area are always worst than the true area and excess, this must be the worst case for approximating a face of R. We will integrate to find the volume of the swept triangle and its excess. Let H be the difference in z-component between t and segment pq. The

length of the segment being swept as a function of h, where $h \in [0, H]$ is $L(h) = \frac{l}{H}h$. Each segment on pqt has its nearest point on the midpoint. The swept area of such a segment is the sum of the areas of the two sub-segments minus a cap at its midpoint. This area is bounded below by the area swept by one of the sub-segments $A_{min}(h) = L^2(h)\frac{\theta}{8}$. Then a lower bound on the swept volume is

$$V_{min} = \int_0^H A_{min}(h)dh = \frac{l^2\theta H}{24}.$$

The excess area generated by each segment is $A_{ex}(h) = \frac{\theta^3}{24}(4r^2 - \frac{L^2(h)}{2})$. So the excess volume generated by triangle pqt is

$$V_{ex} = \int_0^H A_{ex}(h)dh = \frac{\theta^3}{24}H(4r^2 - \frac{l^2}{6}).$$

So, an upper bound on the ratio of excess volume to sweep volume is

$$\frac{V_{ex}}{V_{min}} = \frac{24r^2 - l^2}{6l^2}\theta^2$$

where r is the radius of the robot and l is no smaller than minimum altitude of all input triangles. Hence, the error on the polyhedral approximation of the sweep is $O(\theta^2)$ and produces a close approximation when θ is small.

Additional excess is introduced when we approximate ruled surfaces by a sequence of convex hulls, but because they are only necessary for cap and cup segments, which have length $O(\theta)$, by construction these have $O(\theta^4)$ volume and there are $O(\frac{1}{\theta})$ of them, and so the error they introduce is also $O(\theta^2)$.

4 Path Panning

Suppose we want to navigate between configurations (t_a, θ_a) and (t_b, θ_b). Then θ_a is contained in $[\frac{2\pi i}{n}, \frac{2\pi(i+1)}{n}]$ for some i and t lies in some connected component of S_i. This defines a node that contains (t_a, θ_a) and, similarly, a node that contains (t_b, θ_b).

Given the graph we constructed, we can perform a breadth first search between these nodes to quickly determine whether such a path exists. Since the graph defines an inner approximation of S, we may return a false negative but not a false positive. If a path exists, our search will return a sequence of free space components $\{C_1, C_2, ..., C_n\}$ connecting (t_a, θ_a) and (t_b, θ_b) if one exists.

Given a method of finding paths between two points in the same component, we can construct a valid path from (t_a, θ_a) to (t_b, θ_b) as follows: Navigate inside C_1 from t_a to some point t_2 in $C_1 \cap C_2$. Rotate from θ_a to some θ_2 in the range of C_2. Similarly, for each $i > 2$, navigate inside C_{i-1} from t_{i-1} to some point t_i in $C_{i-1} \cap C_i$ and rotate from θ_{i-1} to some θ_i in the range of C_i. Lastly, navigate in C_n from t_n to t_b and rotate from θ_n to θ_b. The result is a valid path from (t_a, θ_a) to (t_b, θ_b).

For generating a valid path, the choice of t_i is arbitrary. However, a more methodical choice of t_i can reduce path length. Working backwards, we assign t_n to the nearest point in $C_{n-1} \cap C_n$ to t_b and, iteratively, assign t_i to the nearest point in $C_{i-1} \cap C_i$ to t_{i+1}. While the problem is symmetrical, working backwards from the terminal point results in a more intuitive path: the robot will approach an obstacle and then rotate to maneuver around it.

5 Finding Paths in a Free Space Component

The described path planning algorithm requires a method of finding a path between two points in a given component. In general, the problem of finding the shortest path between two points among polyhedral obstacles is NP-Hard [3], so instead we seek a valid and reasonable path. The shortest path can be approximated in polynomial time using a visibility graph [6] [9] [13]. The more densely the graph is constructed, the closer the approximation. However, in practice, achieving a good approximation is expensive.

We instead reduce the problem to finding paths along the surface of the obstacle. The problem of finding shortest paths on the surface is much simpler and can be solved in polynomial time by wavefront propagation [7] or by partitioning of the obstacle surface [16] [11] [12]. We implement a simpler algorithm for finding paths on the surface.

We reduce the problem to finding paths on the surface as follows: To find a path from p to q, which lie in component C, we find all the points where the segment pq intersects with a face of C. If any portion of pq lies outside of C, we replace it with a path on the surface of C. The result is a path that is fairly intuitive: the robot will move directly towards its destination and maneuver around obstacles as it encounters them.

We find a path from s to t on the surface of a triangulated connected component C as follows. First, we use breadth first search to find a sequence of neighboring faces $\{T_1, T_2, ..., T_n\}$ that connect the containing faces of s and t (Fig. 8(a)). Let t_i be the transformation that rotates T_{i+1} about its shared edge with T_i so that the two triangles lie in the same plane. By applying $t_1 \circ t_2 \circ ... \circ t_{i-1}$ to each T_i, the faces are 'unfolded' to all lie in the same plane (Fig. 8(b)). Since the common edges of the triangles are unchanged, the shortest path through all the common edges is the same in the planar problem as it is in the original problem.

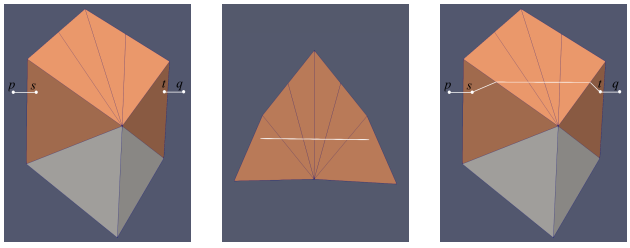
We use a funnel algorithm to find the shortest path

(a) (b) (c)

Figure 8: (a) The sequence of triangles returned by breadth first search for a path from s to t, (b) the sequence unfolded into a 2D and the resulting shortest path, and (c) the 2D shortest path converted back to 3D.

through every common edge of the planar triangles in linear time [4]. We add each common edge, finding the convex hull of the right hand vertices by making only right hand turns and the convex hull of the left hand vertices by making only left hand turns. Whenever the right path becomes left of the left path, move elements from the beginning of the left path to the end of the output path. Whenever the left path becomes right of the right path, move elements from the beginning of the right path to the end of the output path.

Given the shortest path in the planar problem, we find the intersection of the path with each common edge. For each intersection point, we find the point that is the same distance along the equivalent three dimensional edge. The result is the shortest path from s to t through $\{T_1, T_2, ..., T_n\}$ (Fig. 8(c)).



(a) (b) (c)

Figure 9: (a) The sequence of triangles found by going the other way around the vertex shown in Fig. 8, (b) the sequence unfolded into 2D and the resulting shortest path, and (c) the 2D shortest path converted back to 3D.

Since the resulting path is dependent on a sequence of faces $\{T_1, T_2, ..., T_n\}$, we iteratively modify the sequence of faces until we reach a local optimum. For each point where the path goes through a vertex v, the path may improve by going the other way around v. We replace all T_i that are incident on v with faces on the other side of v and perform the two dimensional shortest path algorithm again (Fig. 9). The resulting path is either shorter or unchanged. We repeat this process until the path remains unchanged for every vertex on the path. The result is a locally optimal path on the surface of C.

6 Approximate Maximization of Path Clearance

Given a graph of free space components and the described method of generating paths within a component, we can navigate between any two connected points in the free space. However, the resulting path may contain points on the boundary of the free space, where the robot would scrape against an obstacle. In practice, it is preferable to generate paths with sufficient distance between the robot and any obstacle, for safety and for the maneuverability of the robot.

We expand the graph to account for path clearance by structuring it in levels, where deeper nodes represent free space components with more clearance. The user selects the unit of clearance d and the number of levels l. For each level $l > 0$, S_i^l is the subset of S_i where the robot has at least $d \cdot 2^{l-1}$ clearance from O, generated by taking the Minkowski difference of S_i with a sphere of radius $d \cdot 2^{l-1}$. As before, each component of S_i^l is represented by a node in the graph and edges are placed between intersecting components belonging to neighboring angles ranges. So, at each level, the graph is a representation of the four dimensional free space with clearance of at least $d \cdot 2^{l-1}$.

Additionally, each component is connected to its children: the components of the succeeding level that are contained in it. When a component of S_i^l is narrowed to produce a space with more clearance, it may be eliminated or split into multiple components of S_i^{l+1}. Hence a component may have zero, one, or multiple children.

Given this new graph, we now have the capacity to search for paths in the free space at multiple clearance values. Suppose we want to navigate between components C_a and C_b. A path that traverses deeper nodes corresponds to a path with more obstacle clearance, so we search for a path that maximizes node depth. A simple algorithm to find the deepest path is to visit each node, starting at level 0, and remove the node if doing so does not disconnect C_a and C_b.

A maximal depth search has the advantage of maximizing the clearance on a local basis. The robot can traverse through high clearance components in parts of the path where space is available and also squeeze through tight passages. We choose to increase the clearance unit exponentially in order to capture multiple resolutions with relatively few levels. This results in better paths for problems where the tightness varies greatly at different points in the workspace.

We can further optimize the clearance of the output path by adjusting the paths between two points within a

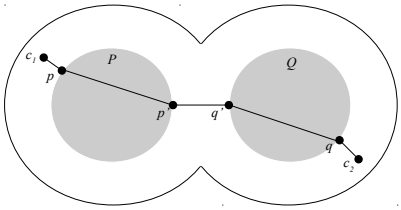


Figure 10: A path from c_1 to c_2 within a component that traverses through its nearest children, P and Q.

component. Suppose we need a path between points c_1 and c_2 within component C, and C has children. Then the children are contained in C and have more clearance. So a path through the children of C is preferable to a path through C. We wish to generate a path though C that avoids the space outside of its children. We first find P and Q, the nearest children to c_1 and c_2, respectively. Next, we find p and q, the nearest point in P to c_1 and nearest point in Q to c_2. If p' and q' are the nearest pair of points between P and Q then $\{c_1, p, p', q', q, c_2\}$ is a path through C that avoids the space outside its children (Fig. 10). If P or Q also have children, we recursively use the same algorithm to find paths in P or Q. This algorithm minimizes the distance the robot must travel in components with low clearance. For faster path finding, we find and store p' and q' for each pair of nodes that share a parent during graph construction.

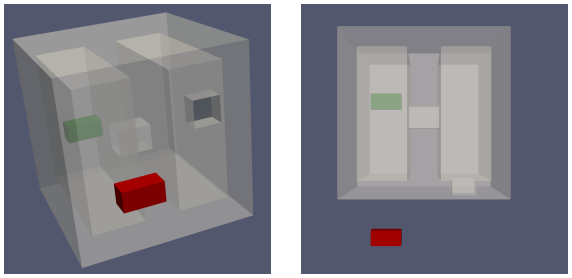
7 Implementation Details

All computation is implemented using the adaptive precision controlled perturbation robustness library [14] [1]. The library ensures results are accurate to the user specified error bound. The Minkowski sums, which are required to generate each three dimensional subspace, are computed using a convolution based approach [8].

For convenience, we save each each subspace and the graph of free space components to files to be used for future path queries. Doing so requires converting the polyhedra with high precision coordinates to meshes with floating point coordinates. The vertices are rounded using a geometric rounding algorithm that preserves the topology of the mesh [10].

8 Results

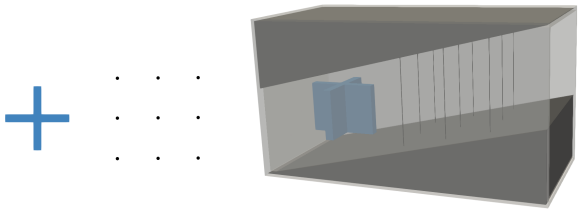
In the problem depicted in Fig. 11, the obstacle is a box with two inner chambers. There are narrow paths connecting the first and second chamber and connecting the second chamber to the outside. The two narrow paths have opposite orientation and navigating from the start position inside the first chamber to the



(a) (b)

Figure 11: Start (green) and end (red) positions for a rectangular robot navigating around a two chamber obstacle.

end position outside requires the robot to make two 90 degree turns. We generate a path for this problem that approximately maximizes the robot's distance from the walls. An animation of that path is available at: `http://web.cs.miami.edu/home/arluck/tworoom/`[2]. The animation shows the scene from three different views: one in each chamber and one outside.



(a) (b)

Figure 12: A cross shaped robot navigating through (a) a 2D integer lattice and (b) a 3D integer lattice of narrow poles on an incline.

Next, consider the problem in Figure 12. There is a cross shaped robot inside a room open on one end. Between the robot and the exit is an integer lattice of narrow poles with a radius of 10^{-5}. This case is difficult for traditional PRM, since it would require a very small step size in order to detect the collision with each pole. We generate a path that rotates back and forth to weave through the lattice while rising to accommodate the incline. An animation of that path is available at: `http://web.cs.miami.edu/home/arluck/lattice/`.

For both problems, we divide the rotation dimension into 40 angle ranges. On one core of a machine with an Intel Xeon E7 CPU, we generate the 40 free spaces in just under 10 minutes for the first problem and 25 for the second problem. We construct the no-clearance graph in about 15 and 30 minutes and each addition level of clearance in 40 minutes and 1 hour. After the one-time computation of the free space, we can generate paths between any two configurations in only 5-10 seconds.

[1] `http://www.cs.miami.edu/home/vjm/robust/`

[2] Animation created by Hal Milenkovic

Acknowledgments

Arluck and Milenkovic are supported by NSF grant CCF-1526335. Sacks is supported by NSF grant CCF-1524455.

References

[1] N. M. Amato, O. B. Bayazit, and L. K. Dale. OBPRM: An obstacle-based PRM for 3D workspaces, 1998.

[2] V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, pages 1018–1023 vol.2, 1999.

[3] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 49–60, Oct 1987.

[4] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, Nov 1987.

[5] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics : The Algorithmic Perspective: The Algorithmic Perspective*, WAFR '98, pages 141–153, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[6] K. Jiang, L. S. Seneviratne, and S. W. E. Earles. Finding the 3D shortest path with visibility graph and minimum potential energy. In *Intelligent Robots and Systems '93, IROS '93. Proceedings of the 1993 IEEE/RSJ International Conference on*, volume 1, pages 679–684 vol.1, Jul 1993.

[7] S. Kapoor. Efficient computation of geodesic shortest paths. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 770–779, New York, NY, USA, 1999. ACM.

[8] M.-H. Kyung, E. Sacks, and V. Milenkovic. Robust polyhedral Minkowski sums with GPU implementation. *Computer-Aided Design*, 6768:48–57, 2015.

[9] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, Oct. 1979.

[10] V. Milenkovic and E. Sacks. Geometric Rounding and Feature Separation in Meshes. *ArXiv e-prints*, May 2018.

[11] D. Mount. On finding shortest paths on convex polyhedra. page 35, 05 1985.

[12] J. O'Rourke, S. Suri, and H. Booth. Shortest paths on polyhedral surfaces. In K. Mehlhorn, editor, *STACS 85*, pages 243–254, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.

[13] C. H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Information Processing Letters*, 20(5):259 – 263, 1985.

[14] E. Sacks and V. Milenkovic. Robust cascading of operations on polyhedra. *Computer-Aided Design*, 46:216–220, Jan. 2014.

[15] F. Schwarzer, M. Saha, and J.-C. Latombe. *Exact Collision Checking of Robot Paths*, pages 25–41. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[16] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 144–153, New York, NY, USA, 1984. ACM.