UNIVERSITY OF MIAMI
DEPARTMENT of COMPUTER SCIENCE

Path Planning in Approximations to 3D and 4D Configuration Spaces

UNIVERSITY OF MIAMI
CENTER for COMPUTATIONAL SCIENCE

Seraphina Gibson, Oberlin College   Dr. Victor Milenkovic, University of Miami
Dr. Elisha Sacks, Purdue University   Chloe Arluck, University of Miami

Introduction

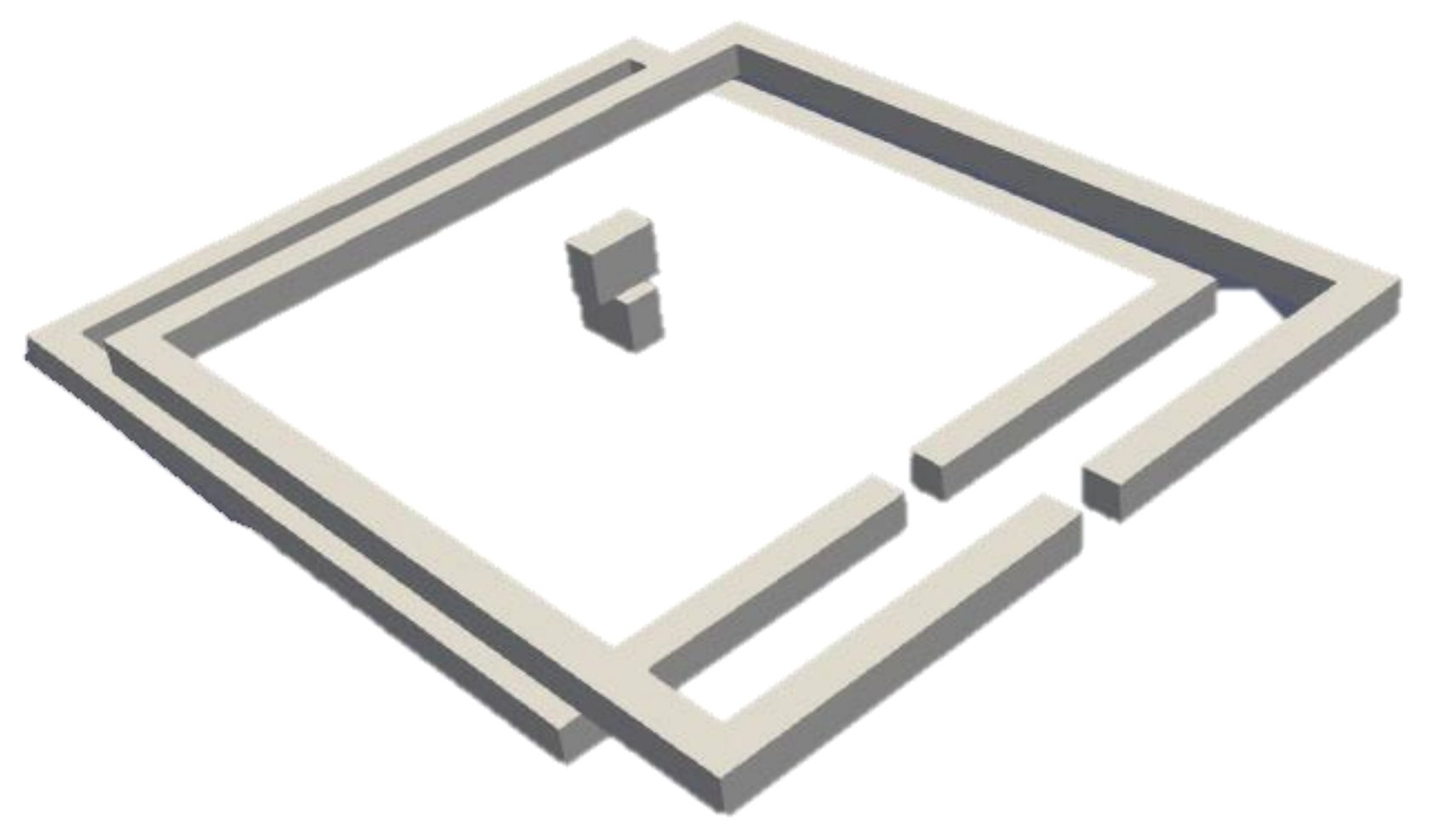
Figure 1: The robot inside it's enclosure. It can fit through the first door, but it's top half is too wide to fit through the second door, so it must turn to make it to the outside. In this version of the problem, the robot can translate in the x and y directions and rotate in the xy-plane, but cannot move vertically.

The path planning problem in 3D space is to find a series of translations and rotations which the moves a "robot" from a given start to a given end without intersecting the obstacles. Applications are primarily in robotics, with different domains putting varied constraints on the problem such as degrees of freedom of the robot and the cost metric by which the paths are judged.

Methods

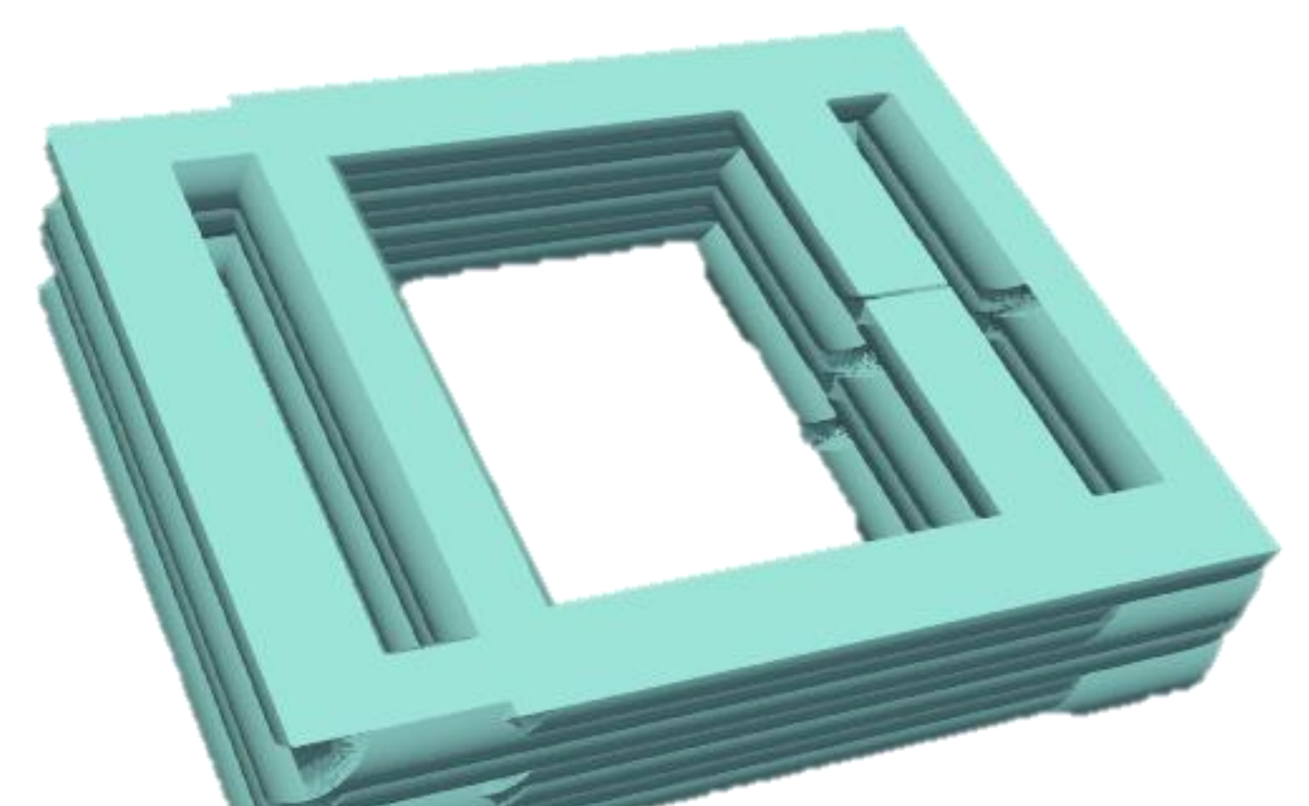
Figure 2 : A slice of the configuration space of the robot in Fig. 1, parameterized by (x, y, θ). The part shown ranges from negative pi to pi. As the structure is periodic in the θ dimension, this range is sufficient to encompass all possible paths.

The space in which the robot and obstacles exist is known as work space. Checking whether the robot has collided with the obstacles in the work space is nontrivial, as they are both complex 3D objects. Thus, the first step in generating our path was to move to *configuration space (c-space),* which is parameterized by (x, y, θ). A single point in c-space represents a translation and rotation of the robot in work space.

The obstacles are translated into configuration space, producing Fig 2. A valid path traversed by a point in c-space is equivalent to a valid path for the robot in work space. As the obstacles in c-space are curved and difficult to work with, they are triangulated to produce a polyhedron. Due to the complex structure in the c-space, this procedure requires care to preserve the necessary structure and avoid issues such as self-intersections.

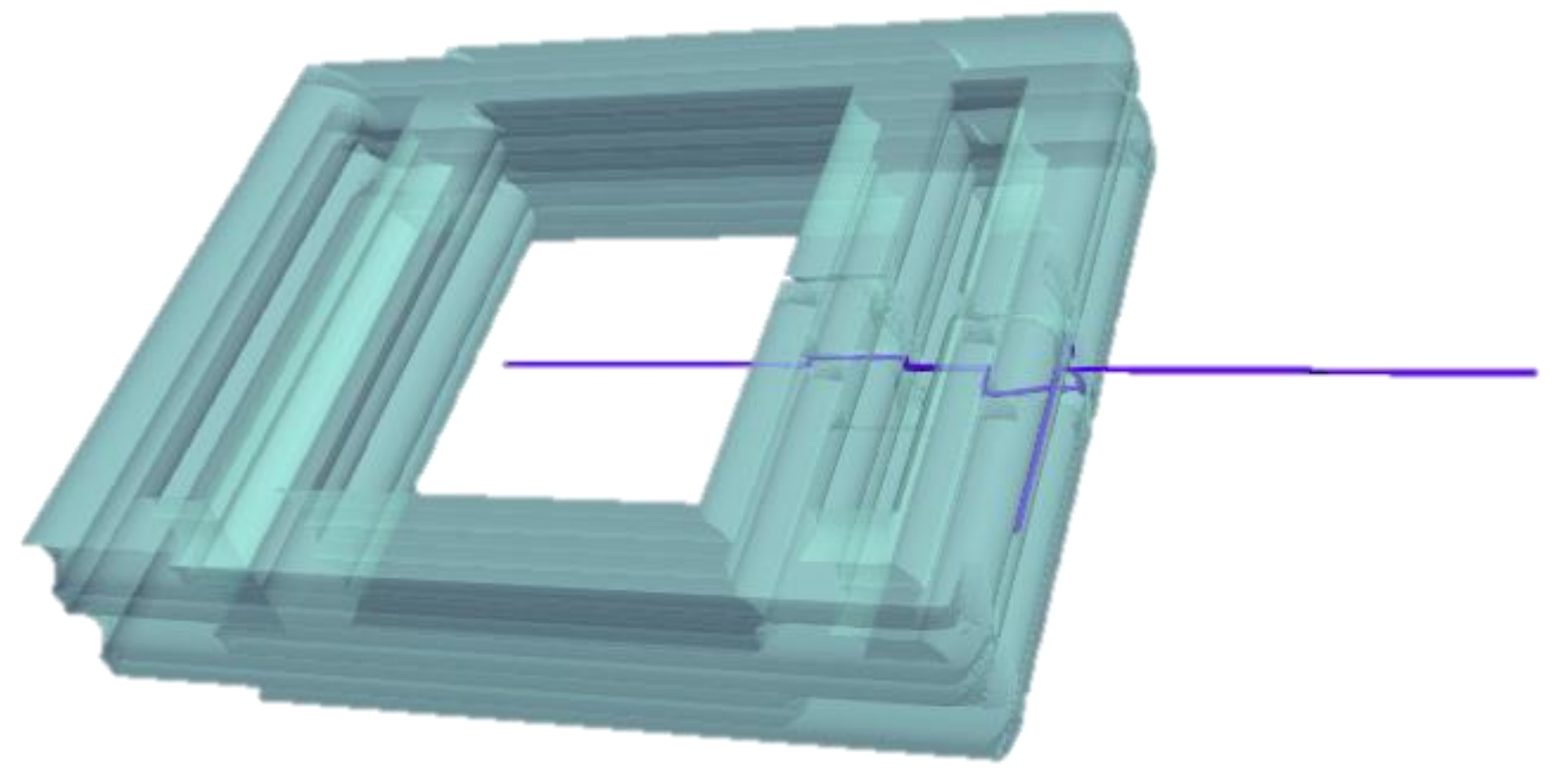
Figure 3: The configuration space, shown partially translucent, with the path generated by the PATHFIND algorithm in bold from the inside of the room to the outside.

Figure 4: A QR code, linked to a video of the path in Fig. 3 animated.

The Algorithm

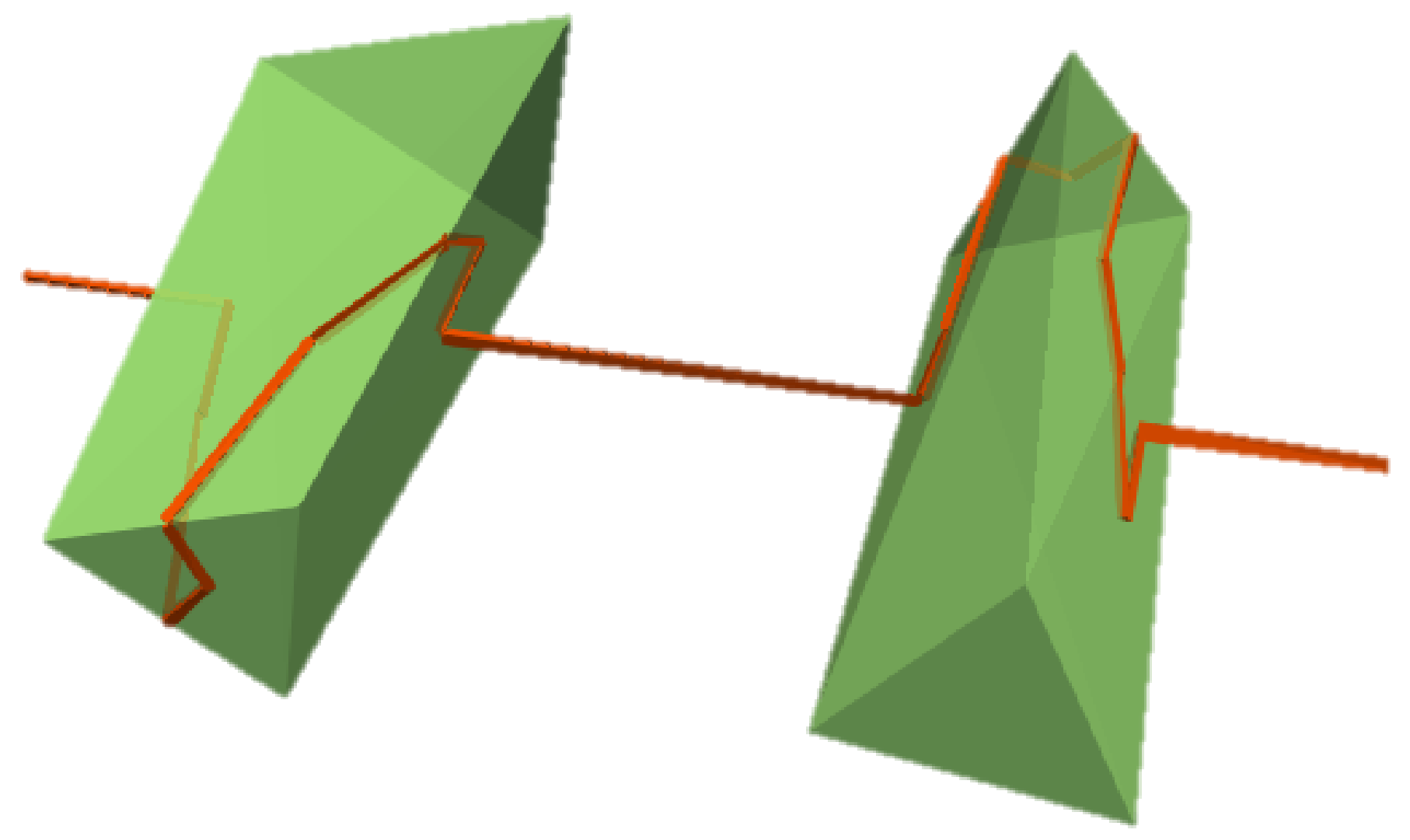
Figure 5: A toy example of two polyhedral figures (green, semi-transparent) and the path (orange) generated from one side of them to the other, to demonstrate the working of the algorithm in a simple case.

PATHFIND

1. Find all intersections with obstacles on the ray from start to end, through brute force, and sort by distance.
2. Move through empty space until an intersection is reached.
3. Breadth-first search over the faces of the intervening polyhedron.
4. Move to the nearest located intersection point with odd index from the start by alternating between centroids of faces and midpoints of edges.
5. Return to step 2 if the end is not yet reached.

Optimal path planning in a 3D space is very computationally expensive, if feasible at all. The focus was instead efficient generation of acceptable paths. A greedy heuristic was used which maximizes time spent on the ray from the start to end, deviating only when an obstacle was encountered.

Much of the work happens in step 4. In step 1, a list of all intersections ordered by their distance from the start is obtained. Those intersections from which progress can be made through empty space are precisely those with odd indices in this (0-indexed) list. Thus the nearest one of these points found by the search in step 3 is the next sub-goal of the algorithm, after which the next segment of the path is always a segment of the ray from start to end.

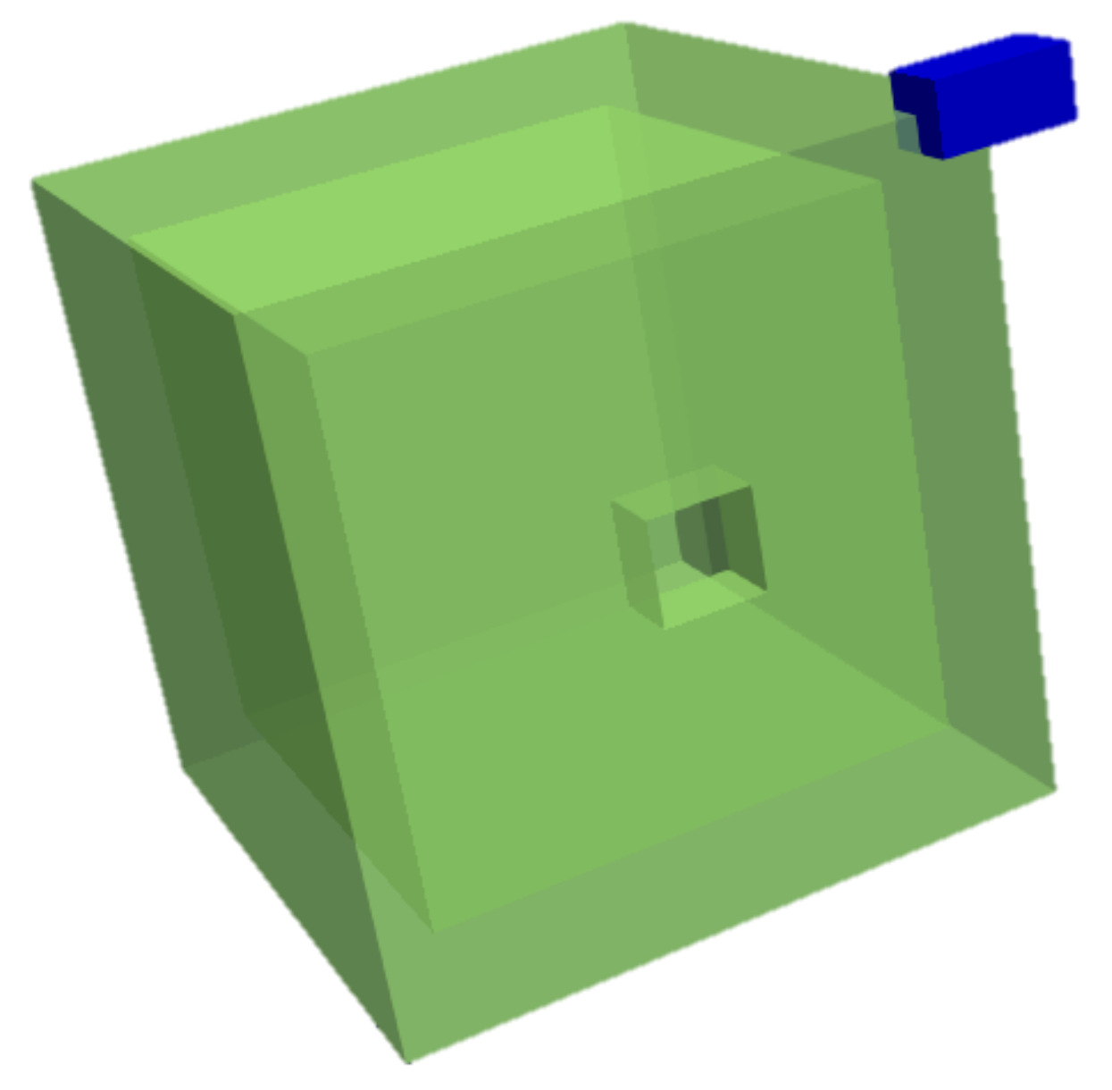
Figure 6: A simple drone (blue) and room (translucent green). The drone's goal is to enter the room, but it is too wide. It must rotate 90 degrees to fit through the small window.

4D Adaptation

A more difficult version of the path planning problem involves a "drone" which can translate in all 3 directions and rotate in the xy-plane. Work space is still 3D, but c-space is now 4D, adding complexity. The approach used was to "smear" the drone over a range of angles, and use the resulting polyhedron to generate 3D slices of the 4D config space at different angles. Due to the smearing, any valid position in a given slice is valid at a range of angles that overlap between nearby slices.

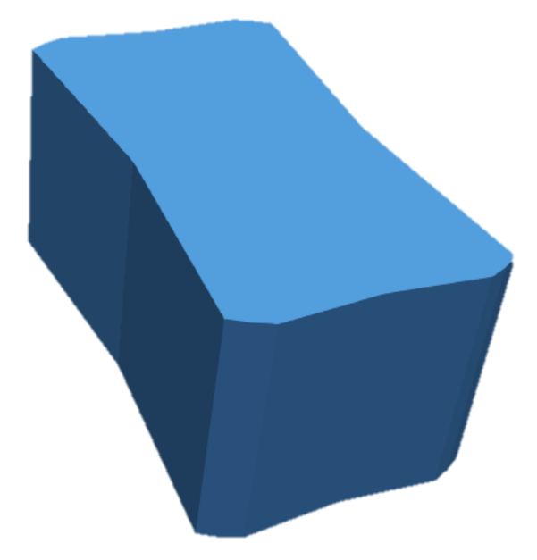
Figure 7: The drone "smeared" through an angle of 2 pi / 40.

A helper algorithm first generates an order to jump between the different slices that can produce a path from start to end, and then PATHFIND produces the sub-paths within each slice. The resulting path alternates between rotations of the drone and translations of the drone, equivalent to changing slices and moving within a slice respectively.

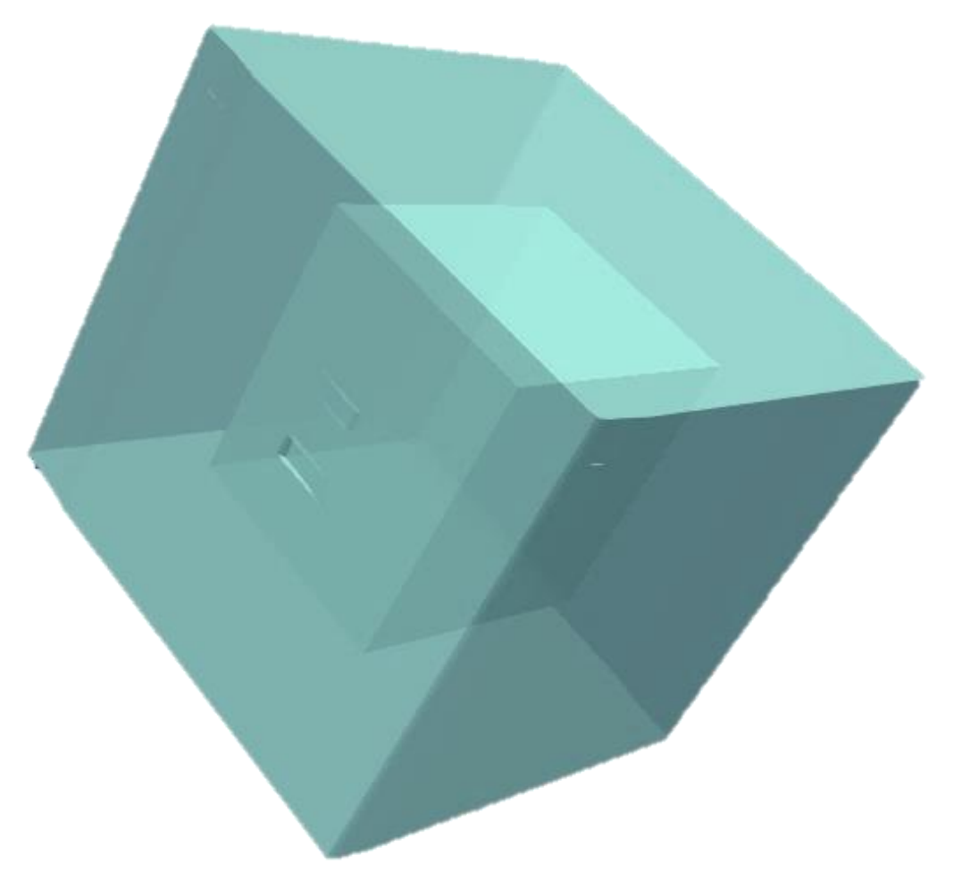
Figure 8: The slice of the c-space centered around the angle of zero. The inside and outside of the room are disconnected because the drone is too wide to fit through the window at this angle.

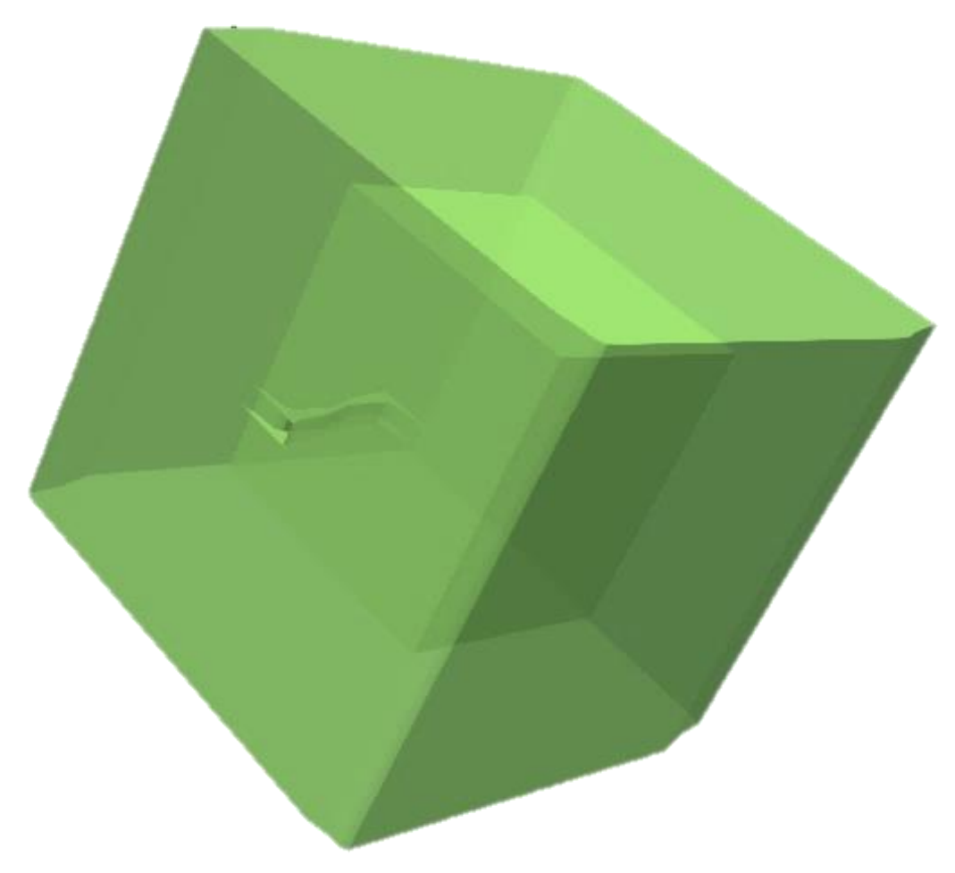
Figure 9: The slice of the c-space centered around an angle of pi/4. A thin channel is now visible: the drone can fly through the window at this angle.

Future Work

The current path generated is guaranteed to be valid, but is not always particularly efficient. One difficulty comes from the complexity of the c-space and the issues with naively triangulating it such as creating self intersections. The more complex method of triangulation often produces very small, or worse, very long and thin triangles, which are not handled well by the algorithm. As a consequence, there are likely significant improvements that can be made by optimizing the path after generating it. One such optimization would be to calculate the gradient of path length with regard to the position of a particular vertex and shift it in that direction, then to repeat this process iteratively over all the vertices multiple times.

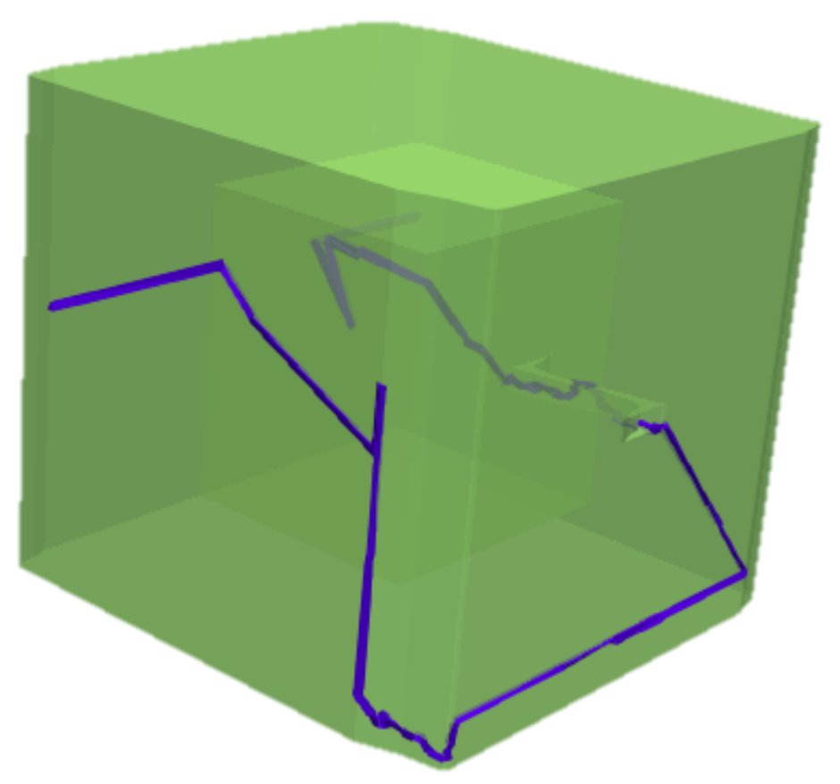
Figure 10: The path (blue) generated in the slice of c-space shown in Fig. 9. This path is a portion of the overall path through 4D configuration space (not pictured).

Acknowledgements

Many thanks to Chloe Arluck, Dr. Elisha Sacks and particularly Dr. Victor Milenkovic for constant support and guidance, and thank you to the program director, Dr. Burt Rosenberg. Thanks goes to Harold Milenkovic for creating the animations. Finally, much appreciation goes to the National Science Foundation, the University of Miami Computer Science Department, and the Center for Computational Science for the funding and support that made the REU possible.

Figure 11: A QR code, linked to a video of the path in Fig. 10 animated.