Path Planning for Orbital Motions

Kimberly Shillcutt[1]

Abstract

Path planning algorithms determine a path from start to goal locations, generally minimizing a cost such as time, distance, fuel, or other parameters. Often, this cost is estimated by the straight-line distance between two points or configurations. However, for a free flying space robot, a straight line is not necessarily the least costly path, because discrete thrusters generate arc-shaped trajectories defined by the current orbit of the robot. So, a way to consider non-straight paths must be found.

This research uses the geometries in Voronoi diagrams to determine and define the path to take for a free flying space robot called AERCam. The current work deals with a two-dimensional world, though the extension to three dimensions is planned. Object-free corridors through which AERCam may safely maneuver are also defined. This work will lead to methods for determining the optimal path.

Introduction

The Autonomous Extravehicular Robotic Camera, or AERCam, is a free-flying space robot intended to fly around the Space Shuttle or International Space Station to inspect areas not visible by astronauts inside, or by remote fixed cameras outside. Tasks which AERCam could perform include the automated inspection of solar panels, detection of anomalies or defects on surface structures, and assistance in human extra-vehicular activities by providing additional camera angles and a mobile light source (Fredrickson, 1997).

AERCam maneuvers by firing nitrogen thrusters arranged about its surface, allowing motion in all directions. Several characteristics of the motions thus generated have implications for path planning. A discrete thrust will produce a curved trajectory in space, instead of the straight line which would be produced on Earth. This trajectory depends on the orbital configuration of the robot and the direction in which the thrusters are fired. Optimizing trajectories for discrete

[1]Graduate Researcher, Robotics Institute, Smith Hall, Carnegie Mellon University, Pittsburgh, PA 15232; E-mail: kimberly@ri.cmu.edu
(research conducted with Metrica, Inc., at NASA Johnson Space Center)

thruster systems has been studied for a flat air-bearing table (Miles and Rock, 1996), but no gravitational effects are present in such a system. Optimizing trajectories on a slanted table, as well as in an orbital environment, adds complications. Moving from point A to point B requires one set of thruster controls, while moving from point B to point A may require a totally different set. In other words, no metric can be defined for which the cost of travel from A to B is always the same as the cost of travel from B to A.

When an object is orbiting the Earth, a discrete thrust produces a change in orbital velocity. A positive change causes the object's orbit to move further away from Earth, while a negative change causes the orbit to move closer to Earth. This radial change is in addition to the movement forward or backward along the direction of the thrust. Combined, these movements create a curved trajectory with respect to another object in approximately the same orbit as the first object, such as the Space Shuttle. Thus, instead of just considering AERCam's relative position to the Space Shuttle, the orbital configuration of the Shuttle must also be known.

Finding an optimal path for a free-flying robot following these trajectories is difficult in one step, so several stages are used to generate an efficient, though non-optimal, path for AERCam. A roadmap based on a generalized Voronoi diagram provides nodes between which the robot can fly. Only the straight line between the nodes is guaranteed to avoid obstacles, however. Therefore, corridors surrounding the path between two such nodes are defined, so that points along a curved trajectory between the two nodes can be tested to see if they also avoid obstacles. A path is then defined as a sequence of nodes, and is chosen by considering the approximate cost of travel between nodes.

This work concentrates on a two-dimensional version of AERCam, which maneuvers on an air-bearing table. This table can be slanted to introduce gravitational variations along different directions, simulating the differences in trajectories that occur when an orbiting robot's thrusters are fired in different directions. Future work is planned to extend these results to three dimensions. The use of a generalized Voronoi diagram simulator, designed by Wade Henning and Frank Hickman at Carnegie Mellon University, aided in the development of this path planner.

Voronoi Diagrams

Voronoi diagrams are based on a distance function, locating the free-space points in an environment which are at equal distances from objects in the environment. In a two dimensional world, all points which are equidistant from two objects are considered part of the generalized Voronoi diagram. Where a third object is also at the same distance, a node, called a three-way meetpoint, is formed. One characteristic of the two-dimensional GVD is that all these nodes are connected, and thus form a one-dimensional roadmap of the environment (O'Dunlaing and Yap, 1985).

The distance to objects in the environment can be found by sensors, or else can be computed if the positions of all objects are known a priori. This second method is used in this work, with all objects defined at the start, and with the distance to each object being the Euclidean distance. The objects are defined as polygons. This requirement is not necessary for sensor-based GVD formation, though modeling real objects by a composite of polygons can be accurate enough, and allows the following analysis to be used.

Path Segment Geometries

The paths between each node are termed generalized Voronoi diagram edges, or path segments. The shape of these path segments depends on which portions of the two nearest objects are the nearest points. These nearest points can be either an object's edge or an object's vertex. Additional nodes are added to the GVD along a path segment whenever one of the two nearest points changes from being an edge to a vertex, or vice versa. Then the path segment shape between each pair of adjacent nodes, both the original three-way meetpoints and the added nodes, can be well described.

Each node is nearest to at least two objects. Each of its adjacent nodes is also nearest to these same two objects (and perhaps another object, as well). Considering these two objects for both nodes surrounding a path segment, the path segment will be one of three types. If both nodes are closest to the same vertex of one of the objects, that is called a point. If the nodes are closest to different parts of an object, that is called an edge. Since there are two objects, there are three possible combinations for the path segment: point-point, point-edge, and edge-edge (see Figure 1).

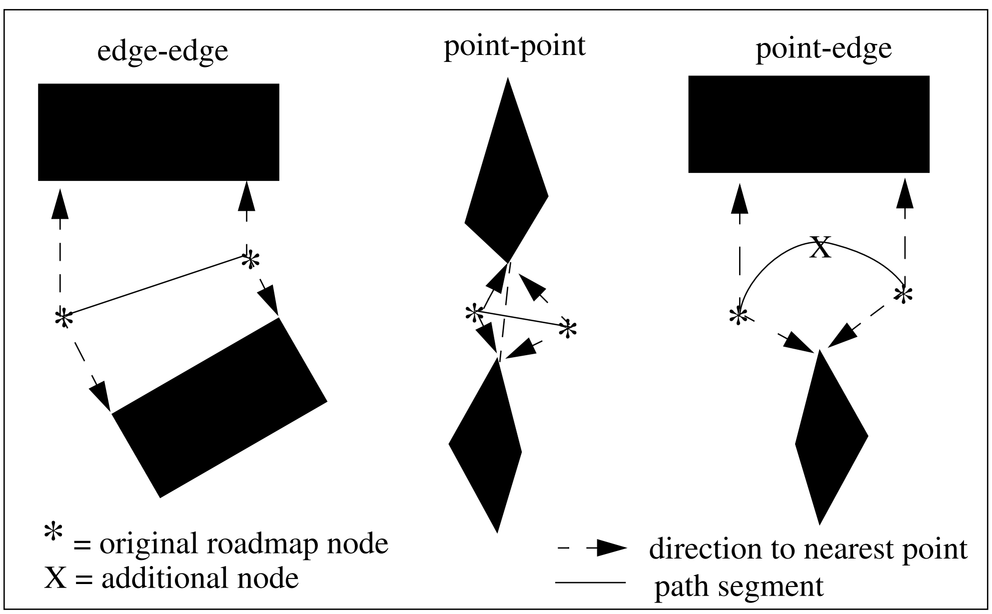

Figure 1. Path Segment Types

The shape of an edge-edge Voronoi path segment is a straight line. The slope of this segment is the average of the slopes of the two object edges. The shape of a point-point Voronoi path segment is also a straight line. This line is normal to the line joining the two object vertices, bisecting that line. A point-edge Voronoi path segment is a parabola segment, concave with respect to the point object's vertex, which is the focus of the parabola. The directrix of the parabola is the edge of the other object. The vertex of the parabola is the parabola point which is closer to the two objects than all other points. This vertex may or may not be within the path segment. If it is, an additional roadmap node is added at this point.

A sequence of these nodes from start to goal is then defined as a path. A straight line between each pair of nodes, when all the nodes in addition to the three-way meetpoints are included, is guaranteed not to intersect an object. This is obvious for edge-edge and point-point path segments. For a point-edge path segment, the parabola segment is object-free by definition, but a straight line between the nodes is also object-free. This can be seen by the fact that the path segment comes from only one half of the parabola. If not true originally, then this will be true once the vertex node is added, splitting the path segment. A straight line between the path segment nodes will be closer to the point object than to the edge object, but never close enough to intersect the point.

Object-Free Corridors

However, trajectories other than straight lines between the nodes in a path are not necessarily object-free. If another trajectory is required, then the trajectory must be tested to ensure that it does not interesect an object. Given a set of path segment nodes, with the distance and direction to the two closest objects for each node, a corridor of free space can be defined. Any point within this corridor is guaranteed to be outside all objects, and accessible by the robot. One type of free-space corridor, for straight line paths between polygonal objects, has been defined by Brooks (1983). For the curving paths of the GVD, a different type of free-space corridor is needed.

Circles utilize the basic characteristic of the GVD, which is that all points in the diagram are equidistant from all closest objects. If the closest object is a given distance away from a Voronoi diagram point, then a circle with a radius equal to that distance is guaranteed to not have any other objects within that area. When the GVD roadmap is generated, the node positions are stored along with all the information required to define such circles.

For each point on the straight line connecting a pair of nodes, the radius of an object-free circle centered on that point can be found. So given some trajectory point not on that line, a point on the line can be chosen, and an object-free circle defined for which the trajectory point can be tested to see if it lies within that circle. The main problem consists of picking the best point on the line, and calculating the radius of the circle for which that given trajectory point should be tested. The circles so determined for the entire trajectory will describe a corridor free of objects.

Picking the point on the line that serves as the circle's origin depends on which of the three types of path segments is being considered. For all these types, the circles centered on the path segment nodes should be considered first. If these circles can confirm that the trajectory point is within the corridor, then further computation is not needed. These node circles are easy to determine, since the radius is just equal to the distance from the node to the nearest objects. This information is readily stored when the roadmap is generated. A circle with a radius equal to this distance will not intersect any objects, so if the trajectory point is less than that radius from the node, it is a valid point.

For point-point segments, an additional circle can be considered, if the trajectory point is not within the node circles. First, determine which object is closer to the trajectory point. Then take the line from that object's vertex that passes through the trajectory point, and find where the line intersects the path segment. If the intersection is within the path segment, that intersection point will be the center of the circle. Otherwise, the trajectory point is not within the corridor, because the node circles did not contain the point either. The radius of the circle can be found by taking the distance from the center to the object vertex. Note that the location of the object vertex can be found by knowing the distance and direction to the object from the nodes, which is information obtained when the roadmap is generated. If the trajectory point is not within this circle, then it is not within the corridor.

For edge-edge segments, as a point on the path segment progresses from one node to the other, the distance of that point from the nearest objects increases or decreases linearly, because the same straight edge of each object is the closest part of the object for the entire segment. So the radius of an object-free circle centered on that point will also linearly increase or decrease. Thus, it is simple to find the radius of such a circle given the two nodes, and the distance of the path segment point from one of the nodes.

To find the path segment point to use as the center of the circle, find which object is closer to the trajectory point, and take the normal to that object edge which passes through the trajectory point. The slope of this object edge can be found from the direction and distance to the objects from each node, which is known from the roadmap generation. The next step is to find the intersection of that normal with the path segment. This intersection point is the center of the object-free circle that should be used to test the trajectory point. If the trajectory point is not within that circle, then it is not within the corridor.

The edge-point segment is the most complicated. This path segment is not linear, but is a portion of a parabola, as described earlier. For purposes of defining the corridor, the straight line between the nodes will be considered the path segment instead of the parabola. Points on this straight line path segment are not equidistant from both objects, but will be closer to the point object in most cases, since the equidistant parabola is concave with respect to that object. So the first step is to find which object is closer to a trajectory point, the edge object or the point object.

If it is closer to the point, then find the normal to the path segment that passes through the trajectory point. The intersection of that normal with the path segment will be the center of the object-free circle to be considered. The distance from that intersection to the object point is the radius of the circle. This is less than the distance to the edge object, but since the trajectory point is closer to the point object anyway, the shorter distance is sufficient.

If the trajectory point is closer to the edge object, then find the normal to this edge that passes through the trajectory point. The intersection point of this normal with the path segment will be the center of the object-free circle. The radius of this circle is the distance from this center point to the point where the normal line intersects the object edge. This distance is greater than the distance to the point object, and so this circle will intersect the point object, but since the trajectory point is closer to the edge, this does not matter. If the trajectory point is within the circle constructed as described, then it is within the corridor.

These corridor defining circles can be constructed with only the information obtained when generating the roadmap, using primarily the underlying basis of the GVD - the distance to nearby objects. To compensate for the width of a non-point robot, the maximum radius of the robot, along with any buffer distance desired, can be subtracted from the circle's radius, ensuring the robot will not hit an object.

Path Selection

In the general case, several paths, or sequences of nodes, will exist for a given start and goal. Selecting which one to use can be done in several ways, one of which is the A* algorithm (Russell and Norvig, 1995). Given a cost of travel from each node to each of its adjacent nodes, this algorithm will find the least cost path from one node to another. Finding the cost to use is the primary difficulty, as this stage of the path planning is based only on geometrical paths, and not time-parametrized trajectories. The straight line distance from one node to another is one option, and will produce relatively good paths in cases where gravitational variations do not produce large effects.

For a slanted air-bearing table, the straight line cost can be modified. For traveling between nodes in the direction of the slant, the entire cost comes from slowing down and stopping the robot at the end node. For traveling in the opposite direction, the cost comes only from the initial thrust. In other directions, the cost depends on the angle alpha between the direction of travel and the direction of the table's slant. The proper function to describe the cost in these situations depends on the amount of slant, the distance being traveled, and the time taken to travel this distance. Since the time can vary for different trajectories, only an approximation of the true cost function was used: the absolute value of the sine of alpha, times the straight-line distance between nodes, times a weighting factor. Further research on how to determine the actual cost should be undertaken. Deciding whether to find a least-fuel trajectory or a least-time trajectory is one additional variable which could be taken into account.

The least cost path determined with these cost estimates may not be the true least cost path from start to goal, when actually enacted. Further optimization can be done, though none is currently implemented in the AERCam path planning code. The original path assumes that AERCam will travel between each node, coming to a complete stop at each. A more optimal trajectory might skip a node or two. With the object-free circles described above, such trajectories can be tested to ensure no collisions occur. If a node can be skipped, producing a savings in cost, then that node can be deleted from the path. Various combinations of adjacent path segments can be tested individually, or candidate path segments can be found by comparing the direction of travel of adjacent segments. If the direction of travel does not change much, a longer, smoother path segment may be possible.

Future Work

The GVD can be extended to three dimensions fairly easily, leading to a one-dimensional roadmap. Work on ensuring the connectedness of such a roadmap is now being done (Choset and Burdick, 1996). Further research needs to be conducted on how to define object-free corridors in three dimensions, or to see if this approach should even be attempted. Generating variously shaped, object-free trajectories between nodes as a first step may be possible, instead of generating straight-line paths and then checking for object intersections of the trajectory deviations from these straight lines. Incorporating the equations for orbital motion, with discrete thrusters, into the path generation may enable such a method.

As mentioned above, a systematic method of determining the most optimal trajectory between a sequence of nodes is not yet developed. Deciding the best velocity at each node is difficult, as there are many possible trajectories which can be generated between any two nodes. In addition, the nodes generated by the Voronoi method may not be the most optimal waypoints for a trajectory. Although the Voronoi nodes may be good starting points for path generation, combining this roadmap with other path planning methods may produce the best results.

References

Brooks, R., "Solving the Find-Path Problem by Good Representation of Free Space," *IEEE Trans. on Systems, Man & Cybernetics*, Vol. SMC-13, no. 2, pp.190-197, Mar-Apr 1983.

Choset, H. and J. Burdick, "Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph," *Workshop on Algorithmic Foundations of Robotics*, 1996.

Fredrickson, S. E., "AERCam II Project Overview," NASA JSC Automation, Robotics and Simulation Division, Internal Publication, May 1997.

Miles, D. W. and S. M. Rock, "Real-Time Dynamic Trajectory Optimization," *Proceedings of the AIAA Guidance, Navigation and Control Conference*, Reston, VA, July 1996.

O'Dunlaing, C. and C. K. Yap, "A 'Retraction' Method for Planning the Motion of a Disc," *Algorithmica*, Vol. 6, pp.104-111, 1985.

Russell, S. and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, NJ, 1995.