SPACECRAFT RENDEZVOUS AND DOCKING WITH REAL-TIME, RANDOMIZED OPTIMIZATION

Jeff M. Phillips[*], Lydia E. Kavraki[†]
Rice University, Department of Computer Science
P.O. Box 1892
Houston, TX 77251-1892

Nazareth Bedrossian[‡]
The Charles Stark Draper Laboratories, Inc.
2200 Space Park Dr, Suite 210
Houston, TX 77058

ABSTRACT

This paper presents a probabilistic approach to solve optimal control problems with application to spacecraft proximity operations. The 6 degree-of-freedom rendezvous and docking problem, using impulsive control, and avoidance of known obstacles and plume impingement is solved. Our solution is then extended to real-time obstacle avoidance. The space is searched by expanding from the start location by applying only feasible controls and coasts, reducing by nearly 50% the variables perturbed in the search. A guided randomized expansion technique explores the search space. A gradient descent approach smooths the path and avoids new obstacles in real-time by "stretching" the best precomputed path in a locally optimal manner.

INTRODUCTION

The ability to service satellites autonomously is ushering in a new revolution in space operations. Future space systems will employ autonomy and high maneuverablity to refuel, reconfigure, and repair satellites. Advanced technology programs such as the NASA Demonstration of Autonomous Rendezvous Technology (DART) program[1] and the DARPA Orbital Express program[2] are now underway to demonstrate the various technologies required to achieve autonomous in-orbit servicing capability.

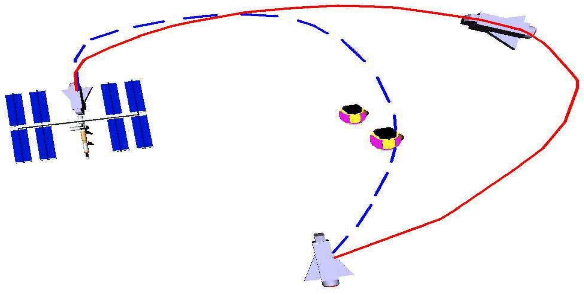

Figure 1: Dynamic obstical avoidance by docking spacecraft

Proximity operations, real-time obstacle avoidance, and plume impingement are some of the issues that need to be addressed in order to achieve the goal of autonomous servicing. The rendezvous control problem has been extensively researched[3]. The plume impingement problem–avoiding contact with harmful jet exhaust–as part of an overall trajectory planner has been dealt with by using collision detection algorithms on truncated cone

[*]Student Researcher, Rice University, Department of Computer Science, Draper Laboratory; jeffp@cs.rice.edu

[†]Associate Professor, Rice University, Departments of Computer Science; kavraki@cs.rice.edu

[‡]Aerospace Systems Group Lead, Draper Laboratory; naz@jsc.draper.com

models of the plume[4] and by using a continuous cost function[5].

In this paper a probabilistic search based approach is applied to solve the rendezvous and docking problem with avoidance of known obstacles and of plume impingement. The paper also describes an approach for real-time obstacle avoidance. The optimal control solution is found through a hybrid offline-and-online technique. In the offline phase a probabilistic search in the spirit of dynamic programming is used to generate the series of controls avoiding known obstacles and plume impingement. In the online phase, a technique based on gradient descent is used to achieve real-time obstacle avoidance. Although errors are not explicitly accounted for, the online phase can serve as an effective feedback control loop to handle error.

The proposed approach is similar to earlier work[5], which also finds a low cost path for an orbiting chase vehicle to dock on an orbiting target vehicle. This earlier approach discretizes the space into a tree network and explores the low cost parts of the network guided by an A* heuristic[6]. However, by limiting the search to a fixed number of possible controls used at every step, this search will miss lower cost paths which occur between or outside these possible controls.

The paper is organized as follows: The second section presents the concept of kinodynamic motion and describes the assumptions under which the orbital rendezvous and docking problem is solved. The third section introduces and motivates the paradigm of probabilistic searching. The fourth section describes the algorithm for Guided Randomized Tree Expansion. The fifth section describes the algorithm for Path Gradient Descent and Dynamic Obstacle Avoidance. In the final two sections the applications and results are explained and the advantages of the proposed system are discussed.

KINODYNAMIC MOTION

Kinodynamic motion planning refers to trajectories planning[7]. A kinodynamic system is described by the static representation (position, orientation, ...) of an object, the derivatives of that representation with respect to time, and the time that the representation of the object occurs. This static representation can be referred to as the *state* of the object. The *configuration* of the object refers to the state, the state's derivatives, and a time. Only the state is required to check for collisions, but in kinodynamic systems, dynamic constraints such as maximum velocity restrict the entire configuration.

Orbital dynamics in rendezvous and docking problems can be effectively modeled with only state and the first derivative of the state in the configuration. The use of the first derivative in the configuration is motivated by the Clohessy-Wiltshire equations[8], which are commonly used to describe orbital dynamics. In this paper, the position and orientation of the chase vehicle determines the state, and the first derivative of the state is referred to as the active part of the configuration.

Control for orbital dynamics can be modeled with instantaneous changes in the active part of the configuration. This impulsive control represents short jet firings by the chase vehicle. A path for the chase vehicle can be described as an array of directional and rotational impulsive controls at specific times, which can be integrated to retrieve the chase vehicle's full configuration at any time. For the orbital rendezvous and docking problem, the solution paths must avoid collisions with the target vehicle and any other obstacles such as debris, must avoid plume impingement, and must minimize fuel use. Avoiding collisions and plume impingement can be accomplished by restrictions on the static state and the controls, so no kinodynamic constraints are directly enforced on the solution path. However, because the fuel use must be minimized, the high velocities are discouraged, and because of orbital drift, minimal velocity bounds exist as well.

PROBABILISTIC SEARCH

In this section a brief overview of probabilistic search is presented as applied to dynamic programming and the spacecraft rendezvous and docking problem. The general approach to solving optimization problems using dynamic programming is to first build a network of waypoints and then search the network for the best path. The conventional approach to waypoint generation is to grid the system state space by discretizing the states and controls. However, this approach may result in many excess waypoints some of which are physically unrealizable. Alternatively, a tree approach can be used where one first seeds the search space with a start and goal configuration, then new waypoints are created by branching off of old ones. Bad edges and bad waypoints are pruned, and the optimal path is harvested from the tree. For tree-based networks, the key to finding optimal paths quickly is in choosing how and when to branch. Probabilistic search uses a weighting function to probabilistically choose which waypoint to expand and then grows the tree from that waypoint by branching outwards based on a random distribution.

In this paper, a probabilistic approach is used to explore the entire, non-discretized configuration space through a randomized process, in the paradigm of Probabilistic Roadmaps[9]. Tree-based exploration is more effective in kinodynamic problems[7], so our approach is more similar to randomized tree expansion planners[10,11,7]. These techniques have been used before to compute autonomous aerial maneuvers[12]. However, conventional tree-based planners have three significant drawbacks for our purposes. (1) In higher dimensional spaces, the dependence on range searching–algorithmic methods for quickly finding points which are close in Euclidean space–makes some problems intractable because of range searching's slow run time for high dimensions. This issue was originally addressed in[13]. (2) Conventional range searching algorithms do not accurately determine "close" points in kinodynamic spaces. A configuration in kinodynamic space is described by a vector of parameters which are dependent on one another. Range searching determines "closeness" using a Euclidean metric which works on the 1-norm of the configuration vector. This metric assumes the parameters of the configuration vector are independent, which is not the case in kinodynamic systems such as ours. (3) Because these traditional methods bias their search expansions in such a way as to maximize the region reachable, they also may bias away from low cost paths. Results from these techniques have empirically been shown[7] to generate "straight" or "smooth" paths in position space, but these traits do not necessarily equate to low cost. The method introduced in this paper, Guided Randomized Tree Expansion, addresses these shortcomings.

It should be noted that our modification of standard randomized tree-based planners was inspired by work in[13] which solved planning problems for an extremely high-dimensional system with a nonconventional method also using a randomized tree-based planner.

This paper also explores the capability to further refine the initial path. Elastic Band techniques [14,15] refine preexisting paths by following cost gradients of waypoints on the path. Recently, this technique has been extended to low-dimension kinodynamic problems[16]. Elastic Band techniques rely on a closed-form cost gradient for waypoints on the path. This paper describes a variation on Elastic Band techniques to a higher dimensional kinodynamic problems. In particular, the proposed approach not only avoids obstacles as in previous work, but it also minimizes path cost by a metric other than distance and smoothness. The variation of the Elastic Band technique avoids obstacles in real time while preserving low cost.

GUIDED RANDOMIZED TREE EXPANSION

The goal of guided randomized tree expansion is to explore the nondiscretized configuration space with a focus on the low energy regions. This will find a path more quickly by focusing the search and return a path that not only avoids obstacles, but also has a low cost. The algorithm is modeled on randomized tree-based exploration techniques and A* network searches.

Algorithm

The inputs to the algorithm are: (1) a start and a goal configuration of the chase vehicle relative to the target vehicle; (2) a method for checking whether a particular configuration of the chase vehicle is in collision with the target vehicle or violates some other constraint; (3) a method to integrate forward the effects of an impulsive control; (4) a metric to determine cost for an impulsive control; and (5) an estimate of the cost from a particular configuration to the goal configuration. The user can specify either how many iterations the algorithm can run, N, or a threshold for total cost of the path that a successful algorithm must be under. The pseudocode is given in Algorithm 1.

Algorithm 1 Randomized Tree Expansion

```
1: for $i = 0$ to $N$ do
2:     $p$ = CHOOSE_WAYPOINT()
3:     $n$ = EXPAND_WAYPOINT($p$)
4:     ADD_TO_TREE($p$,$n$)
5: end for
```

The tree is seeded with the start configuration and is then built by incrementally choosing a waypoint in the tree using the **CHOOSE_WAYPOINT** method, expanding the waypoint to a reasonable new waypoint using the **EXPAND_WAYPOINT** method, and then adding the new waypoint to the tree, if possible, using the **ADD_TO_TREE** method which also attempts to connect the new waypoint to the goal configuration. The program terminates when it has iterated N times at which point it returns the lowest cost complete path found or null if no complete path to the goal has been found, or it can terminate early if a complete path has less than a specified cost. Experiments have shown that halting the algorithm when a specified cost threshold is reached returns a path more quickly, but running for N iterations finds lower cost paths when more search time is available.

The **ADD_TO_TREE** method serves four purposes. (1) It checks to see whether the path between waypoints p and n is feasible. This step uses the input to the algorithm which checks for collisions or any other constraints. (2) If the path is feasible, the method calculates the cost to reach the new waypoint, n, by summing the total cost to get to the previous waypoint, p, and the cost determined by the metric for cost based on the path from p to n. This is stored with the new waypoint. (3) The method then adds the new waypoint, n, to the tree by adding it as an outbranch from the previous waypoint p. (4) Finally, an estimated cost to the goal configuration is calculated. This cost is also stored in the waypoint. If this estimate is less than some threshold and the new waypoint is within some threshold for Euclidean distance to the goal, then the method checks if it can connect the new waypoint to the goal configuration using the same given function it used to verify the connection from the previous waypoint to the new waypoint. If the new waypoint, n, connects to the goal, then the cost for the total path is calculated, and if this is the lowest cost path calculated so far, it is stored to be returned later.

The **CHOOSE_WAYPOINT** method determines which waypoint will be expanded. In conventional probabilistic tree-expansion techniques[11], waypoints are chosen from a random distribution based on proximity to other waypoints with the intent of exploring the space. In potential field methods[17], waypoints are generally chosen by looking at the most recently generated waypoint, which, if expanded to correctly, should also be the waypoint with the lowest total cost. Potential fields do not explore the space very well because they tend to follow a single branch, but they seek low cost solutions. Our method for choosing waypoints combines both of these aspects. We compute a weight for each waypoint and then select a waypoint randomly with a probability proportional to its weight. This can effectively and quickly be done using a heap data structure. The weight is inversely proportional to a function of the *A*_cost*, inversely proportional to a function of the *out_degree*, and proportional to a function of the *order* in which it was generated.

$$weight(p) = \frac{order(p)}{A^*\_cost(p) \cdot out\_degree(p)} \qquad (1)$$

,The *A*_cost* is the sum of the total cost calculated to reach the waypoint and the estimated cost to reach the goal. The A* cost, which predicts the overall cost, is widely used in graph searching to improve search times and serves to focus the search towards waypoints which will more likely lead to a low energy path. The *out_degree* represents the number of times the waypoint has been expanded and effectively limits the number of times the same waypoint is chosen. By weighting waypoints proportionally to the *order* in which they are picked, more recent nodes tend to be chosen more frequently. This encourages the tree to explore the space faster.

By choosing waypoints randomly but based on the above distribution, the algorithm is not limiting the search simply to following the locally most optimal direction. Exploring in locally non-optimal directions as well prevents the search from stalling or getting stuck in local minima. Consider a search technique that exclusively uses some energy function, where the global minimum is the goal, in order to guide the search to the goal–for example naive potential field planners. Now consider a configuration which is an energy local minimum, for instance a high energy region is between it and the goal. A planner strictly following the locally low energy gradient may drift into this local minimum then requiring an exploration method, such as a random walk, to escape the local minimum. Controlled random deviation from the local gradient avoids this. For example, weighting the choice of which waypoint to expand with the *A*_cost* reasonably focuses the search towards the low energy and feasible direction, but by not strictly following the best *A*_cost*, the search can find the globally optimal path.

The **EXPAND_WAYPOINT** method produces a new waypoint, n, which can likely be connected to the previous waypoint, p. To adhere to constraints of motion and to ensure that connections can be achieved with a simple discrete control, we generate the new waypoint, n, by applying a control to the previous waypoint, p, and integrating over a time. To explore the space, the control direction and magnitude as well as the time to integrate are chosen randomly from a range[10]. By expanding the waypoint when applying a control, the cost can be easily calculated based on the amount of control, and only feasible movements from a waypoint need be considered and explored.

Including Range Searching

Alternatively, Guided Randomized Trees can be generated by including information about the number of nearby points. This can be incorporated into the weight function. Randomized tree expansion algorithms[10,7] rely heavily on the number of nearby points because this heuristic can ensure that the randomized exploration of the space has high likelihood of expanding into the entire space and that it does not spend too much time searching in the same area. By penalizing configurations which are somehow close to each other, the tree will tend to branch off of the configurations on the perimeter of explored space and thus expand into unexplored regions of the configurations space. This is a very desirable characteristic, especially if the path to the goal is narrow, and thus hard to find.

However, the way in which closeness should be measured in configuration spaces where the motion between configurations is restricted by kinodynamic constraints is not entirely clear. Traditionally, a 1-norm of the vector of numbers representing the configuration is used, but this distance function is not reliable in systems where certain terms of the state vector are dependent upon other terms. Two different ways of measuring distance are proposed as well as a way to quickly search them.

The first alternative distance function attempts to measure the closeness of the second configurations to the region where the first configuration can expand into. The function returns the control cost for the first configuration to exactly reach the second configuration. The second alternative compares the regions in which two configurations can expand into. It first lets the second configuration drift with no control for a set time. It then returns the control cost for the first configuration to reach the drifted second configuration.

Most conventional techniques to find all near neighbors use properties of a 1-norm distance function. So, they will not work for these alternatives. Metric trees[18] instead use only properties of a metric distance function. It builds a binary tree of the data by splitting on a data point. All configurations within a median distance are placed in the inside sub-metric tree, and the rest are placed in an outside sub-metric tree. The split is continued recursively. When retrieving near-neighbors of a configuration, the triangle inequality and symmetric properties of the metric distance function are used to, if possible, either rule out any near-neighbors from being on the inside sub-tree or from being on the outside sub-tree. This allows lookup to avoid comparing all the configurations and instead examines on the order of a logarithm of the number of configurations.

Unfortunately, since both alternative distance functions are only near metrics–they slightly violate the triangle inequality and are not symmetric–using metric trees does not always retrieve 100% of the nearby configurations. Experimental data shows that metric trees return between 70% and 100% of the nearby configurations for first alternative distance function, and between 85% and 100% of the configurations for the second alternative distance function. The effect of the missed near-neighbors with metric trees was qualitatively analyzed with a 3D visualizer. The pattern of tree growth could not be distinguished between the use of metric trees and the use of a brute force technique which calculates the distance function for ever pair of configurations. However, the use of metric trees was asymptotically faster.

By incorporating the number of near-neighbor configurations in the weight function decreased the importance of the terms in the weight function concerning the order-generated and out-degree of the configurations. However, this added term is expensive to calculate. With an optimization that creates separate metric trees for different time buckets, calculating the near-neighbors term, increases by 3 to 4 the time to generate similarly large trees.

Comparison to Other Path Planners

Guided Randomized Tree Expansion solves the set of problems where a large configurations space with kinodynamic constraints needs to be searched and a low cost path needs to be extracted. A similar randomized path planning algorithm Probabilistic Road Maps[9] also will in general rapidly search a large configurations space, but this technique generally is not well-equipped for solving problems with kinodynamic constraints. The points generated are completely random within the configurations space and thus are not garaunteed to connect together, especially not with a low cost. After generating many paths (which may or may not violate constraints), genetic algorithms[6] could be run on the paths to attempt to breed constraint-abiding and near-optimal paths; this is known as Adriadne's Clew Algorithm [19]. However, paths created by breeding two paths have small likelihood of not violating constraints and of improving on cost. So, this technique could take a hopelessly long time. Dynamic Programming using A* optimization of a discretized search space [5] will also return a constraint satisfying low cost path. To search large configurations spaces quickly requires that the discretization of the possible expansions from a node be rather coarse. This will results in a poor picture of the cost function over the configurations space. To increase this coarse discretization will increase by an exponential factor the number of nodes required to be searched. While this technique with unlimited space will eventually find a solution, Guided Randomized Tree Expansion will quickly find a solution, then it can continue to build on the old solution to find better solutions. At any point (after the first solution is quickly found), the best path found so far can always be returned, and the algorithm can continue to run, searching for new and better paths.

This Guided Randomized Tree expansion algorithm is meant to be run offline, and deterministic completeness cannot be guaranteed. A notion of probabilistic completeness–where the probability that algorithm will solve the query goes to 1 as the time it has run increases–is discussed in terms of similar algorithms[10,20]. But Guided Randomized Tree Expansion does not fall under this analysis. The algorithm can be run many times simultaneously in parallel and the low cost path from all the runs can be returned. On each run of the program, a specific series of controls describing a solution path may occur with nonzero probability. A proper analysis of this technique lies outside the scope of this paper. In practice the technique finds paths often and quickly for this problem. This will be discussed with more detail in the results section.

PATH GRADIENT DESCENT

After the best path is extracted from the Guided Randomized Tree Expansion algorithm, it can be further refined by calculating the cost gradient of the path, and perturbing the path in that direction incrementally. An entire path has many degrees of freedom, but by analyzing each waypoint on the path individually and following the cost gradient of that waypoint, the approach becomes manageable. This part of the process can be run offline.

Path Representation

A path is stored as an array of waypoints where each waypoint describes a configuration. In our case a configuration represents position, velocity, orientation, rate of change of orientation, and time. The controls applied at each waypoint can be stored as well. The end points of the path are fixed, but all intermediate waypoints are variable. The algorithm deforms the intermediate waypoints in an attempt to reduce cost while avoiding collisions and other forbidden behavior.

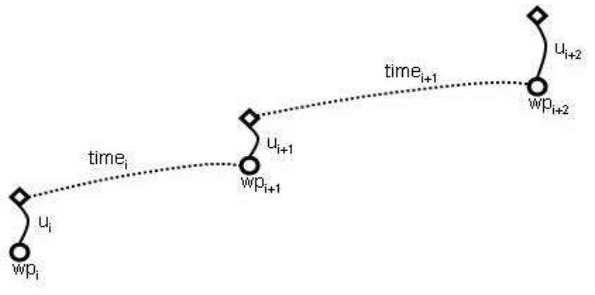

Figure 2: Path segment

Figure 2 shows a segment of a path where the circles are waypoints, *wp*, the solid lines, *u*, represent impulsive controls instantaneously altering the active part of the configuration at the waypoint, and the dashed lines are the integration of the change in configuration over time. With the ability to integrate forward in time, a path can be described by the controls applied and the time elapsed between controls. Thus the configurations at the waypoints can be derived as the result of integration.

Cost Function

For such a path representation the cost of the path can be written in terms of controls and elapsed time, even if it depends on the configurations, since the configurations can be derived. The objectives are to minimize cost in terms of the controls and to avoid obstacles. The algorithm does not vary the elapsed time between waypoints. The cost function for a single waypoint, such as wp_{i+1} in Figure 2, is:

$$\mathbf{COST}_{wp_{i+1}} = \begin{array}{c} avoid(obstacles, wp_{i+1}(u_i))+ \\ control(u_i) \end{array} \quad (2)$$

COST is the sum of a function, *avoid*, which repels waypoints from obstacles and a function, *control*, which reduces cost of controls.

Functions *avoid* and *control* are designed for a system where the equations to integrate position over time can be written in a linear form. This is also motivated by the Clohessy-Wiltshire (CW) equations[8]. The kinodynamic system is broken into two parts: the static part, position and orientation, represented by x; and the active part, velocity and rate of change of orientation, represented by $\dot{x}$. The ϕ matrix, which transforms an initial configuration to a final configuration, is a function of time. An abstraction of a linear kinodynamics system can be written:

$$\begin{bmatrix} x_f \\ \dot{x_f} \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{bmatrix} \begin{bmatrix} x_i \\ \dot{x_i}. \end{bmatrix} \quad (3)$$

The *control* function needs to take into account the control applied at the particular waypoint and at the adjacent waypoints. Consider the waypoint labeled wp_{i+1} in Figure 2. If the configuration at waypoint wp_{i+1} is perturbed, the controls u_i and u_{i+1} must be altered. But altering controls applied at waypoints wp_i and wp_{i+1} will not necessarily integrate the function to wp_{i+2}, so additional control must be applied at that third waypoint. First perturb u_i and then solve for u_{i+1} in terms of the static part of the configuration at wp_{i+2}. This will align the static part of the integrated system with wp_{i+2}. Then u_{i+2} can be adjusted to compensate for the difference in the active part. Once all controls are written in terms of u_i, the gradient can be found with respect to u_i:

$$\begin{aligned} u_{i+1} &= \phi_{12}^{-1}(x_{i+2} - \phi_{11}x_{i+1}) - \dot{x}_{i+1} \\ x_{i+1} &= \phi_{11}x_i + \phi_{12}(\dot{x}_i + u_i) \\ \dot{x}_{i+1} &= \phi_{21}x_i + \phi_{22}(\dot{x}_i + u_i) \\ u_{i+2} &= \dot{x}_{i+2} + u_{i+2'} - \phi_{21}x_{i+1} - \phi_{22}(\dot{x}_{i+1} + u_{i+1}). \end{aligned} \quad (4)$$

The *avoid* function is inversely proportional to the distance between the obstacles, O, and the state of the chase vehicle, squared. The state of the chase vehicle can be calculated as described above also in terms of u_i and compared to the position of the obstacles, O, to easily determine distance. This is equivalent to the repulsive external forces used in Ref.[15].

$$\begin{aligned} avoid(O, u_i) &= \tfrac{1}{(O - x_{i+1}(u_i))^2} \\ x_{i+1} &= \phi_{11}x_i + \phi_{12}(\dot{x}_i + u_i) \end{aligned} \quad (5)$$

Algorithm

To minimize the cost of the path, the gradient of each waypoint in the path is independently calculated and followed for some short distance ϵ. This process is iterated N times.

Algorithm 2 Path Gradient Descent

```
1: for $i = 0$ to $N$ do
2:     randomly permute order of waypoint considered
3:     for $j = 0$ to # waypoints do
4:         calculate gradient
5:         follow gradient $\epsilon$
6:     end for
7: end for
```

Algorithm 2 provides pseudocode for path gradient descent. Randomly permuting the order in which the gradient descent is run on the waypoints ensures that there is no bias in how the path deforms based on the order in which the waypoints are minimized. This process will quickly converge to the local minimum cost of the path.

Optimizations will cause the system to converge more rapidly. (1) Waypoints with higher gradients can be followed more often or for a larger value of ϵ, while waypoints with smaller gradients can be followed less often, or for a smaller value of ϵ. This adaptive refinement will not only allow the process to work faster, but will also allow it to refine more specifically. (2) In regions where two controls are applied in close time proximity to each other, they can potentially be combined into one control. This may reduce cost and may be more practical to execute. Conversely, in regions where there is a long drift without a control, a control can potentially be added. Although this could make the overall maneuver more complicated, it could possibly reduce the cost.

Dynamic Obstacle Avoidance

The path gradient descent technique can be extended to work in a changing environment. Because the gradient is recalculated at every iteration, updating information about current position and movement of obstacles will not affect the process. As a result, this technique can be used as a control feedback system to account for error in integrating control and for dynamic obstacle avoidance. To account for dynamic obstacles, a prediction can be made for where they will be at future times. This prediction can be compared to the planned path for the chase vehicle, and the chase vehicle's trajectory can be modified to avoid possible collisions with the predicted position of the obstacle. As the prediction is refined, the trajectory of the chase vehicle to avoid collision can be refined in a control feedback loop.

Since path gradient descent is an iterative process that converges to a minimum, the longer it is run, the lower the cost will become. This results in a tradeoff between autonomy and control. Adaptively, this method can quickly avoid obstacles that suddenly are on a collision course with the path and then reconverge to a low cost path. After the first iteration of the program, a collision free path can be returned and immediately used, and this collision free path can be continued to be refined until the locally optimal path of the new environment is found.

EXPERIMENTS

In a series of experiments, Guided Randomized Tree Expansion and Path Gradient Descent were applied to the rendezvous and docking problem of two orbiting vehicles. These techniques could easily and effectively be extended to other kinodynamic systems.

Orbital Dynamics

The translational dynamics are governed by the Clohessy-Wiltshire (CW) equations[8] describing changes in position and velocity of the chase vehicle relative to the target. The CW equations can be easily manipulated to provide a good cost estimate [21] for use in A^* cost calculations. The rotational dynamics are modelled assuming constant orbital rate. Tests were run on docking problems from about 2000 feet away from the target vehicle for a system representing the space shuttle as the chase vehicle and the space station as the target vehicle. The chase vehicle is assumed to make impulsive changes in velocity and angular rate bounded by user specified thresholds. The results are compared to a Bang-off-Bang (BOB) estimate[21] where the cost of the maneuver is calculated using the CW equations. This estimate, applies a single control from the start configuration, drifts without additional control to the goal configuration, and applies a final control to match the goal configuration's velocity and angular rate. The BOB estimate ignores collision and plume impingement.

Plume Impingement

Plume can be thought of as the exhaust resulting from jet firings of the chase vehicle and is harmful to the target vehicle. The plume is modelled as teardrop-shaped collision objects that appear when control is applied in the opposite direction. The chase vehicle is modelled as having jets capable of firing along its x, y, and z axis in its coordinate frame, although, these could be modelled differently. To calculate the direction of the plume, the control applied is translated into the chase vehicle's coordinate frame. The size of the plume clouds is determined by the component of the control along each of the x, y, and z axes. If a collision with the target vehicle is detected, the configuration is rejected, just as if the chase vehicle collided with the target vehicle. All collision detection is done with RAPID[22]. Figure 3 shows plume impingement on the target vehicle.

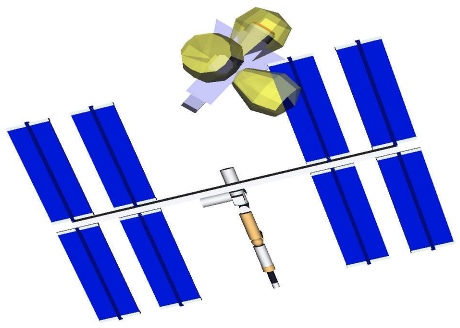

Figure 3: Plume Impingement

Randomized A* Tree Expansion Tests

Tests were run on AMD Athelon 1GHz processor computers. 10,000 trees were generated with the Guided Randomized Tree Expansion algorithm. Each tree was built with 40,000 branches in about 2 seconds. This takes a total of about 10 minutes on a 32-node cluster. The low cost path was reported from each tree, compared to the BOB estimate, and plotted in the chart in Figure 4. The results are grouped in categories 10% of BOB wide. So the bar labeled 100 on the x axis represents the number of trees which generated a path between 100% and 110% of the BOB estimate. The lowest reported cost was 71% of the BOB estimate, although most minimum costs were about twice the BOB estimate.

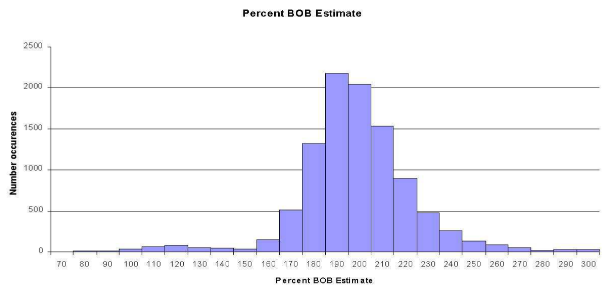

Figure 4: Best costs for 10,000 Randomized A* Tree Expansion trees

An example tree and the minimum path for that tree with plume clouds are displayed in Figures 5 and 6. The example proximity operations depicted started with the velocity and angular rate of the chase vehicle matching that of the target vehicle. The goal state for the chase vehicle also matches the target vehicle's velocity and angular rate. The tree and paths displayed represent the three-dimensional position of the chase vehicle in the relative frame of the target vehicle. The velocity, orientation, and angular rate are difficult to display in a stationary two-dimensional image. Through the use of a three-dimensional simulation model visualizer we developed, orientation can be displayed by the orientation of the spacecraft drawn at different waypoints on the tree or path. By interpolating between waypoints of the tree and the path, the velocity and angular rate can be visualized.

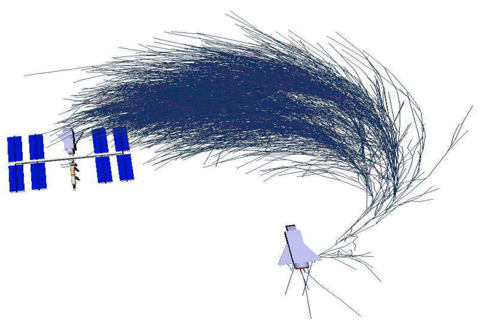

Figure 5: Guided Randomized Expansion Tree

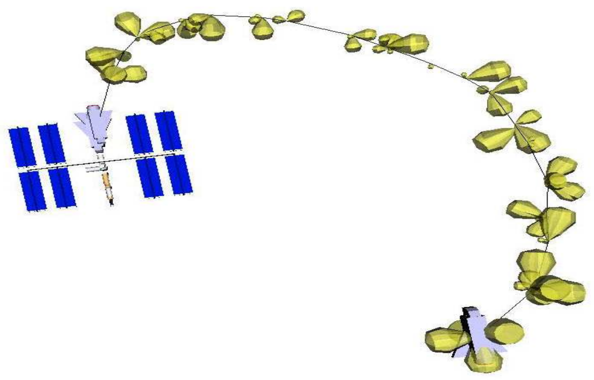

Figure 6: A low cost path with plume clouds

Path Gradient Descent Tests

Tests for path gradient descent were run on a PIII 450Mhz computer. Precomputed paths can be smoothed by distributing impulse throughout the path. The gradient descent would converge to about 200% of the BOB estimate within 5 seconds, and would tend to completely converge to between 50% and 66% of the BOB estimate with more time. Figure 7 shows the smoothing results of gradient descent. The initial path is dashed and the refined path is continuous.

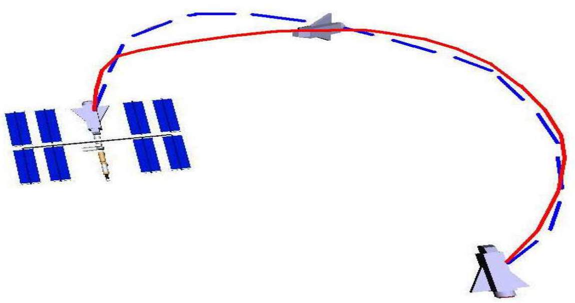

Figure 7: Path Gradient Descent

Dynamic Obstacle Avoidance Tests

Tests for dynamic obstacle avoidance were run on a PIII 450 MHz computer. Moving obstacles (representing asteroids) were introduced to the system such that they would exactly intersect the path at the specific time to cause a collision with the chase vehicle. The path deforms to avoid the obstacle after the first iteration in about 1 millisecond. However, this results in an extremely high cost path. After about 1 second, the path would converge to about 300% BOB estimate and eventually to about 100% BOB estimate. These results of course depend on how the introduced obstacles interfere with the optimal path. In our test cases, we intentionally introduced obstacles to dramatically interfere with the optimal path representing the worst possible cases. Of course, if the introduced obstacles are far from the optimal path, then they will have minimal effect on how the path changes and the reconvergence will not take much time. The image in Figure 8 displays, with thin lines, the state of the path as it avoids the obstacles and then relaxes. The original path is dashed and the final path is continuous and bold. Notice how the path initially jumps very far from the obstacles and then reconverges to a smoother, lower cost path. Also note that even though the path initially has sharp vertices indicating high cost controls near the obstacle, the early part of the refined path is very similar to the path which is eventually converged to. So, the new path which avoids obstacles can immediately be followed and then refined as the chase vehicle is beginning to move in the right direction away from the obstacle. And since this is modeling a 2000 second operation, waiting a couple seconds for a fully reconverged path is still a small fraction of the total maneuver time.

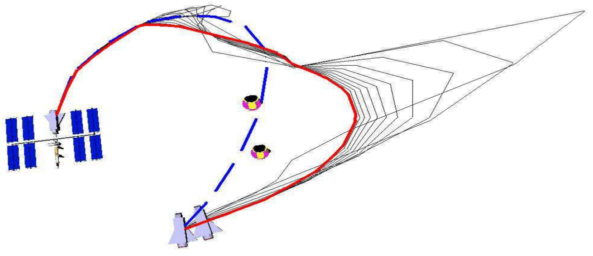
Figure 8: Dynamic Obstacle Avoidance in motion

DISCUSSION

We believe our methodology can be easily extended for use in other kinodynamic systems. The technique has the advantage of finding a minimum cost path while avoiding obstacles.

The Guided Randomized Tree Expansion technique deviates from standard randomized tree expansion techniques[11,7] in two ways. (1) It incorporates a cost function that is used as a guide. (2) It does not necessarily use any range searching. In general path finding applications, most time is spent in collision checking. Our model has very simple and quick collision checking so our speed bottleneck was in range searching. As a result a technique was devised that works without range searching. However, including range searching in the CHOOSE_WAYPOINT function helps expand the tree more effectively and consistently. The use of the added term may have advantages in configuration landscapes with more obstacles.

The goal in developing the Guided Randomized Tree Expansion technique was to increase the rate of convergence to the goal and to find lower cost paths compared to standard randomized tree expansion techniques. We accomplish this by focusing the search to the part of the state space that would likely contain a low cost path if there were no obstacles. This slows the convergence of the exploration of the complete space but increases the convergence of the exploration of the space likely to harbor the best path.

In our EXPAND_WAYPOINT method for the Guided Randomized Tree Expansion, controls are sampled with which to expand randomly from a uniform distribution. A different distribution, such as a normal distribution, may have effective results. This should be explored further.

By choosing branches based on controls[10] that can be integrated to determine movement instead of a perturbation of all variables in the configuration, only feasible solutions are generated and the variables altered when exploring the search space are reduced by nearly 50%.

The methodology relies on linearly modeled discrete impulsive controls. But many systems for which this technique might be applied are better modeled with continuous control and/or nonlinear control. Future work should explore expanding the Guided Randomized Tree Expansion and the Path Gradient Descent to systems with continuous control and nonlinear control.

ACKNOWLEDGEMENTS

We would like to thank Andrew Ladd for many great discussions on the tree expansion algorithm used in the paper. Work on this paper by Lydia Kavraki has been supported in part by NSF 9702288, NSF 0114796, a Whitaker Biomedical Engineering Grant and a Sloan Fellowship to Lydia Kavraki. Work on this paper by Jeff Phillips has been supported by Draper Laboratories and by NSF 9702288 REU.

References

1. NASA-http://nmp.jpl.nasa.gov/st6/TECHNOLOGY/ rendezvous_tech.html .

2. DARPA-http://www.darpa.mil/tto/programs/astro.html .

3. Kluever, C. *Feedback Control For Spacecraft Rednzvous and Docking. Journal of Guidance, Control, and Dynamics* **22**(4), 609–611 (1999).

4. Vaugn, R. M., Bergmann, E. V., and Walker, B. K. *Collision Detection for Spacecraft Proximity Operations. Journal of Guidance, Control, and Dynamics* , 225–229 Mar-Apr (1991).

5. Jackson, M. C. *A Six Degree of Freedom, Plume-Fuel Optimal Trajectory Planner for Spacecraft Proximity Operations Using an A* Node Search.* Master's thesis, MIT, (1994).

6. Russel, S. and Norvig, P. Prentice Hall (2002).

7. LaValle, S. M. and Kuffner, J. J. *Randomized Kinodynamic Planning. International Journal of Robotics Research* **20**(5), 378–400 May (2001).

8. Clohessy, W. H. and Wiltshire, R. S. *Terminal Guidance System for Satellite Rendezvous. Journal of the Aerospace Sciences* **27**, 653–658 September (1960).

9. Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. *Probabilistic Roadmaps for Path Planning in High Dimensional Configuration Spaces. IEEE International Transactions on Robotics and Automation* **12**(4), 566–580 June (1996).

10. Hsu, D., Kindel, R., Latombe, J.-C., and Rock, S. *Randomized Kinodynamic Motion Planning with Moving Obstacles. International Workshop on Algorithmic Foudations of Robotics* (2000).

11. Hsu, D., Latombe, J.-C., and Motwani, R. *Path Planning in Expansive Configuration Spaces. International Journal of Computational Geometry and Applications* **9**(4-5), 495–512 (1999).

12. Fazzoli, E., Dahleh, M. A., and Feron, E. *Robust Hybrid Control for Autonomous Vehicle Motion Planning.* Technical Report LIDS-P-2468, Laboratory for Information and Descisions Systems, MIT, (1999).

13. Ladd, A. M. and Kavraki, L. E. *Using Motion Planning for Knot Untangling. International Workshop on Algorithmic Foudations of Robotics* (2002).

14. Quinlan, S. and Khatib, O. *Elastic Bands: Connecting Path Planning and Control. IEEE International Conference on Robotics and Automation* **2**, 802–807 (1993).

15. Brock, O. and Khatib, O. *Elastic Strips: Real-Time Path Modification for Mobile Manipulation. International Journal of Robotics Research* , 5–13 (1998).

16. Lamiraux, F. and Bonnafous, D. *Reactive Trajectory Deformation for Nonholonomic Systems: Applications to Mobile Robots. IEEE International Conference on Robotics and Automation* , 3099–3104 (2002).

17. Barraquand, J., Langlois, B., and Latombe, J.-C. *Numerical Potential Field Techniques. IEEE Transactions Systems, Man, Cybernetics.* **22**(2), 224–241 (1992).

18. Ciaccia, P., Patella, M., and Zezula, P. *the VLDB Journal* (1997).

19. Ahuactzin, J. M., Talbi, E.-G., Bessiere, P., and Mazer, E. *10th European Conference in Artificial Intelligence* (1992).

20. Ladd, A. and Kavraki, L. E. *Generalizing the Analysis of PRM. IEEE International Conference on Robotics and Automation* (2002).

21. Wie, B. In *Space Vehicle Dynamics and Control*, 282–285. AIAA Education Series (1998).

22. Gottschalk, S., M. C, L., and Manocha, D. *OBBTree: A Hierarchical Structure for Rapid Interference Detection. SIGGRAPH* , 171–180 (1996).