Vehicle Motion Planning with Time-Varying Constraints

W. Todd Cerven[1], Francesco Bullo[2], and Victoria L. Coverstone[3]

University of Illinois at Urbana-Champaign

Introduction

With the growing emphasis on vehicle autonomy, the problem of planning a trajectory in an environment with obstacles has become increasingly important. This task has been of particular interest to roboticists and computer scientists, whose primary focus is on kinematic motion planning [1]. Typical kinematic planning methods fall into two main categories, roadmap methods and incremental search methods, both of which find collision-free paths in the state space. Roadmap methods generate and traverse a graph of collision-free connecting paths spanning the state space, while incremental search methods, including dynamic programming [2] and potential field methods [3], perform an iterative search to connect the initial and goal states. For the purely geometric path planning problem, deterministic algorithms have been created that are *complete*, i.e., they will find a solution if and only if one exists. Unfortunately, these suffer from high computational costs which are exponential in system degrees of freedom. This cost has motivated the development of iterative randomized path planning algorithms that are *probabilistically complete*, i.e., if a feasible path exists, the probability of finding a path from the initial to final conditions converges to one as the number of iterations goes to infinity. The introduction of the Rapidly-exploring Random Trees (RRTs) of LaValle and Kuffner [4] allowed both for computationally efficient exploration of a complicated space as well as incorporation of system dynamics. The RRT grows a tree of feasible trajectories from the initial condition, or root node. Each node, or waypoint, on the tree represents a system state and has possible trajectories branching from it. Through use of an embedded planning routine, the tree incrementally builds itself in random direc-

[1]Senior Member of Technical Staff, currently at The Aerospace Corporation, 15049 Conference Center Drive, Suite 1029, Chantilly, VA 20151, Member AIAA

[2]Assistant Professor, Coordinated Sciences Laboratory, 1308 West Main Street, Urbana, IL 61801

[3]Associate Professor, Department of Aerospace Engineering, 104 South Wright Street, Urbana, IL 61801, Associate Fellow AIAA

tions, node by node, until the final conditions are met (within accuracy bounds). Frazzoli [5] demonstrated that a hybrid systems representation of vehicle dynamics, when coupled with the RRT, could be used to address moving obstacles and time-invariant final conditions in a real-time environment. This paper presents a similar approach which provides probabilistic completeness in the presence of both time-varying obstacles and final conditions while using a simpler algorithmic procedure. In addition, a novel approach to provide error mitigation of the embedded planner in a hybrid system-based RRT is presented. An example is then given in which the proposed algorithm is applied to the landing of a spacecraft on an idealized asteroid.

An RRT-based Approach

Background

The idea of this method is to incrementally build a *tree* of feasible trajectories to efficiently explore a reachable space, where a *tree* is a directed graph in which all nodes (excepting the root) have one parent node and an unspecified number of child nodes. The basic RRT algorithm [4] can be seen in Figure 1(a).

The original RRT algorithm, shown to be probabilistically complete, extended the tree by picking the closest (Euclidean metric ρ) node on the tree to the random point and choosing the best constant input from a finite predetermined set. For this simplistic embedded planner, the system equations of motion are propagated according to the input for a predetermined time. If no collisions are found, a new child node is added to the tree corresponding to the propagated state.

Frazzoli adapted this method for control of autonomous vehicles where motion is represented by a concatenation of motion primitives [5]. He redefined the metric ρ as the cost-to-go function of an optimal control problem [2] and replaced the constant input set with an online planning algorithm that found solutions for the obstacle-free planning problem. Using the *Extend* routine in Figure 1(b), he was able to address moving obstacles by looping through successively close nodes on the tree until one was found which allowed an acceptable trajectory. If a collision free trajectory was found, the routine would grow the tree and then try to connect to the final state. A completeness proof of this approach is in [5]. A limitation on this method lies in its inability to address time-varying final conditions.

A New Approach

We use a primitive-based hybrid system model that expands upon that in [5], where the dynamics of the system, commonly described using an ordinary differential equation, are instead modeled by computed state flows in response to differing inputs. These can be separated into two types of primitives, *reference trajectories* and *maneuvers*, where reference trajectories are precomputed trajectories with a variable time duration within a given interval and maneuvers are fixed time primitives connecting reference trajectories. This is a hybrid system in that the state space is defined by the finite set of trajectory primitives, the continuous space over which each primitive can be applied (i.e., where the dynamics of the system are invariant), and time. The control variables then consist of the reference trajectory time durations and the parameters defining each maneuver. Initial and final conditions are assumed to lie on reference trajectories. A more detailed description of this system is found in [6]. Furthermore, we assume that there exists an embedded planner guaranteed to find motion planning solutions in an obstacle-free environment subject to time-varying final constraints and an upper time limit. The RRT-based algorithm for this approach follows the general loop in Figure 1(a), calling the *Extend* routine in Figure 1(c) to try to connect the trajectories represented by the tree to a state defined by the function *RandomState*. Once again, the metric is defined as the cost-to-go function, but the state variable x now includes time as well. This addition both accommodates a time-dependent final condition and alleviates the need for cycling through the nodes in the tree to address time-varying obstacles. In this *Extend* routine, *NearestNeighbor* merely finds the closest node in the tree to x instead of sorting the tree nodes and cycling through them. *InputFound* then calls the embedded planner to find a feasible trajectory from the nearest node state to x and *NoCollision* incrementally checks the resulting trajectory for collisions. Provided a collision-free trajectory is found, the tree is extended by *AddChildren*. Unlike the approach of Frazzoli, this allows for the addition of single or multiple nested child nodes along the reference trajectories. *Extend* will repeat this process with the last created node until x is reached or a collision-free trajectory cannot be obtained. Once *Extend* exits, *BuildRRT* keeps looping until a solution is found. Probabilistic completeness for this algorithm can then be shown as follows:

Lemma *Assuming no two RRT milestones lie within a specified $\epsilon > 0$ of one another for the given metric, this method is probabilistically complete.*

Proof: Noting that the appropriate input is always generated by the online planning algorithm if it exists and is assigned a specific nonzero execution time, it follows from Theorem 3 of [4] that this method is probabilistically complete. ■

Embedded Planner Error Mitigation

The embedded planner naturally has a prescribed accuracy and, as a result, an error that can be propagated as trajectories are concatenated together. Although this effect could be troublesome, the framework of the RRT also allows for correction of errors from the underlying planning algorithm. For the initial incarnation of the RRT [4], a constant input was chosen from a finite set and was highly unlikely to control the system to the intended final state. As a result, the actual final state (as found by integrating the system under the input) is that which is stored as the new node state rather than the targeted state. Thus, replanning from that node takes into account the error correction. When addressing the hybrid system, the characteristics of these errors become important, as replanning can only occur from nodes on reference trajectories. Thus, when integrating along the trajectory, nodes would only be added to the tree where the state matched the trajectory primitive within an acceptable error. While this methodology is useful in practice, it is not a complete error correction, as the correction is only the projection of the total error onto the trajectory primitive. It is notable that, although limitations of handling the error in this manner were not quantified, this method was shown to be consistently effective for the example problem.

Asteroid Landing Example

The example described here is that of a spacecraft landing on a celestial body similar to the asteroid Ida. We made simplifying assumptions by geometrically modeling the asteroid as a 60 km long cylinder with radius 12 km and modeling the gravity as a Newtonian point source. While it is known that the gravity field about a non-spherical body is more complicated than the point source model, it is used here for simplicity. The algorithm could then be extended to other gravity models when needed. The setup of the problem is seen in Figure 2, where the initial position of the spacecraft is 18 km above the surface of the asteroid with the final condition of "landing" at a point just off the surface on the other side of the asteroid. Although this problem could have been cast in a rotating reference frame, an inertial reference frame was chosen to show algorithm performance in relation to time-varying obstacles and final conditions. Additionally, artificial constraints are imposed to limit motion to Ida's plane of rotation and an annulus with radii between 19.1 and 38.3 km of the center of mass. An upper bound of 46 hours was placed on the transfer time. The embedded planner used in the *InputFound* routine was based on a dynamic programming approach and can be found in [6]. Note that, rather than sample the reachable set, which is computationally impractical to define, the sampling in the function *RandomState* was done over free hybrid state space. Here, the reference trajectories in the hybrid system are defined as the set of all circular orbits and the maneuvers are defined through an online local planning algorithm. From a start point at $[x^T, v^T]^T = [0, 30, 0, -0.0062, 0, 0]^T$ (km, km/s), the randomized algorithm grows a tree as seen in Figure 2, the result of 11 randomized planner iterations. The results of these iterations are given in Table 1.

The "Greedy Loops" column in the table refers to the fact that the error correction mentioned in Section can cause a loop in the *Extend* routine. This occurred in step 11, where the *Extend* algorithm looped an extra time to reach the inertial final state. The overall solution was computed in 64.2 seconds, with a final state of $[-26.2, -12, 0, 0.0026, -0.0057, 0]^T$ (km, km/s) at 12.3 hours. This is an error of .03 percent in position and .2 percent in velocity from the final condition.

Of course, this is a planner based on randomized methods, and, as such, every solution to this algorithm will be slightly different with different run times. A batch of 50 runs of the aforementioned example were completed and yielded a median run time was 95.9 seconds (on an 850 MHz Pentium III computer) with 78% of the cases taking less than 200 seconds. The median number of randomized planner iterations was 11, with 90% taking less than 40 iterations. In every case tested, the algorithm successfully converged to a solution.

Conclusion

This paper presented a new variant of the Rapidly-exploring Random Tree (RRT) for use with a motion primitive-based planner. By including time as a state it is able to accommodate time-varying obstacles and final conditions. This method is shown to be probabilistically complete, finding a solution with a probability of one as the number of iterations goes to infinity. This method was then applied to the example of a spacecraft landing on an idealized asteroid, for which analysis of a batch of runs was completed. This method showed itself to be reliable with typical run times of less than 3 minutes. While the randomized method shown is not optimal, there exist methods to refine the tree to increase optimality.

Acknowledgments

This research is supported in part by the U.S. Army Research Office under grant DAAD 190110716, and by NSF grant IIS-0118146. The authors also gratefully acknowledge Professor Emilio Frazzoli as well as Professor Steven LaValle and his research group, whose comments were beneficial in the development of this paper.

References

[1] Latombe, J.-C., “Motion Planning: A Journey of Robots, Molecules, Digital Actors, and Other Artifacts,” *International Journal of Robotics Research*, Vol. 18, No. 11, 1999, pp. 1119–1128.

[2] Bertsekas, D. P., *Dynamic Programming and Optimal Control, Vol. 1*, Athena Scientific, Belmont, MA, 2nd ed., 2001, Chap. 1, pp. 18–34.

[3] Sundar, S. and Shiller, Z., "Optimal Obstacle Avoidance Based on the Hamilton-Jacobi-Bellman Equation," *IEEE Transactions on Automatic Control*, Vol. 13, No. 2, 1997, pp. 305–310.

[4] LaValle, S. M. and Kuffner, J. J., "Randomized Kinodynamic Planning," *International Journal of Robotics Research*, Vol. 20, No. 5, 2001, pp. 378–400.

[5] Frazzoli, E., Daleh, M. A., and Feron, E., "Real-time motion planning for agile autonomous vehicles," *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 25, No. 1, 2002, pp. 116–129.

[6] Cerven, W. T., *Efficient Hierarchical Global Motion Planning for Autonomous Vehicles*, Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, IL, Oct. 2003.

List of Table Captions

Iteration	End Condition	Greedy Loops
0	Collision	0
1	Not Reachable	0
2	Not Reachable	0
3	Collision	0
4	Not Reachable	0
5	Not Reachable	0
6	Collision	1
7	Collision	1
8	Not Reachable	0
9	Not Reachable	0
10	Collision	1
11	Connected	2

Table 1:

List of Figure Captions

Figure 1: Algorithm Pseudocode, (a) (top) Basic RRT loop, (b) (left) Frazzoli *Extend* routine, and (c) (right) Proposed *Extend* routine; ρ is a predefined metric.

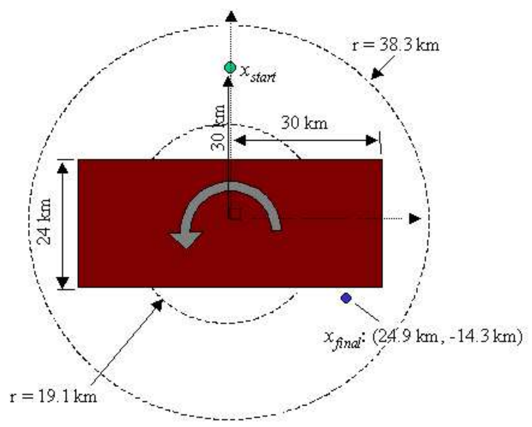
Figure 2: Ida Landing, setup and relative target location(left), tree and final path in rotating reference frame(right)

```
BuildRRT($x_{start}$,$x_{final}$)
1: tree.Initialize($x_{start}$)
2: for iterations = 1 to maxiterations
3:     $x_{rand} \leftarrow$ RandomState()
4:     Extend(tree,$x_{rand}$,$x_{final}$)
5:     if $\rho(x_{rand}, x_{final}) < \epsilon$ break loop
6: return tree
```

```
Extend(tree,x,$x_{final}$)
1: for all $x_{near}$ in SortedNodeList(tree,x)
2:     if InputFound($x_{near}, x, u$)
         and NoCollision($x_{near}, x, u$)
3:         tree.AddChild($x_{near}, x, u$)
4:         if $\rho(x, x_{final}) < \epsilon$ return success
5:         else let $x_{near} = x$, $x = x_{final}$ and goto 2:
6: return failure
```

```
Extend(tree,x,$x_{final}$)
1: $x_{near} \leftarrow$ NearestNeighbor(tree,x)
2: if InputFound($x_{near}, x, u$)
     and NoCollision($x_{near}, x, u$)
3:     tree.AddChildren($x_{near}, x, u$)
4:     if $\rho(x, x_{final}) < \epsilon$ return connected
5:     else let $x_{near} = x$, $x = x_{final}$ and goto 2:
6: return not reachable or collision
```

Figure 1:


r = 38.3 km
x_{start}
30 km
30 km
24 km
x_{final}: (24.9 km, -14.3 km)
r = 19.1 km

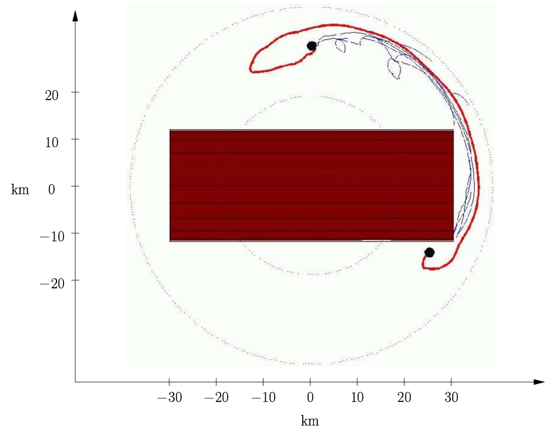

20
10
km 0
−10
−20
−30 −20 −10 0 10 20 30
km

Figure 2: