Intelligent Computation of Reachability Sets for Space Missions

Erik Komendera[1] and **Daniel Scheeres**[2] and **Elizabeth Bradley**[1]
[1] Department of Computer Science
[2] Department of Aerospace Engineering
University of Colorado at Boulder

Abstract

This paper introduces a new technique for intelligently exploring the *reachability set* of a spacecraft: the set of trajectories from a given initial condition that are possible under a specified range of control actions. The high dimension of this problem and the nonlinear nature of gravitational interactions make the geometry of these sets complicated, hard to compute, and all but impossible to visualize. Currently, exploration of a problem's state space is done heuristically, based on previously identified solutions. This potentially misses out on improved mission design solutions that are not close to previous approaches. The goal of the work described here is to map out reachability sets automatically. This would not only aid human mission planners, but also allow a spacecraft to determine its own course without input from Earth-based controllers. Brute-force approaches to this are computationally prohibitive, so one must focus the effort on regions that are of interest: where neighboring trajectories diverge quickly, for instance, or come close to a body that the spacecraft is orbiting. In this paper, we focus on the first of those two criteria; the goal is to identify regions in the system's state space where small changes have large effects—or vice versa—and concentrate the computational mesh accordingly.

Introduction

Trajectories for space missions require extensive and careful planning due to limited fuel budgets and the complexity of gravitational dynamics. For many applications, planners decompose the problem into smaller segments and solve each one using a simplified version of the dynamics. This works reasonably well for maneuvers such as gravity assists, but it fails in systems with highly irregular bodies, such as the 243 Ida-Dactyl system, which have rapid timescales and strong nonlinearities. To plan trajectories in systems like this, one must model the full nonlinear dynamics—and do so 'on the fly,' in order to handle unpredictable effects like perturbations.

An important concept in space mission planning is the *reachability set*: the set of all states $\vec{x}$ that can be reached in some prescribed time interval Δt from a given initial condition $\vec{x}_0(t_0)$, under the operational constraints of the spacecraft. Accurate reachability sets are incredibly important, not only for planning orbits, but also in determining impact and escape scenarios. They are, however, very complicated objects. A spacecraft's state space has six dimensions—three positions and three velocities—and as the craft moves under the gravitational influence of the surrounding bodies, the state evolves nonlinearly (and often chaotically) through that space. An impulse from the craft's engine, called ΔV, can be treated as an instantaneous jump in the velocity subset of this space. If the engine can fire in any direction, the space of possible results of such an action is a solid three-dimensional sphere—the "ΔVsphere"—around the state-space point where the burn was applied. (If the engine is limited to burns in the orbital plane, the space of possible results is a two-dimensional "ΔVdisk".) To compute the reachability set, one must track the evolution of that ΔVsphere over time. That is, the Δt reachability set for a single burn of magnitude $\Delta V$from an initial condition $\vec{x}_0(t_0)$ is the three-dimensional volume embedded in six-dimensional state space that is traced out by the ΔVsphere as it dynamically evolves over the period Δt .

In space missions, Δt might represent a mission requirement: an arrival deadline, for instance, or the lifetime of the spacecraft. Even if Δt is small, reachability sets are hard to compute because the nonlinear dynamics of gravitation quickly cause the 3D ΔVsphere to deform into a complicated volume in the 6D state space. The dimension and complexity make it all but impossible for humans to visualize these sets, which has become a serious hurdle in mission planning. In the absence of systematic and computationally tractable approaches to perform full mappings of the state space, usual practice is to rely on intuition, simplified models for motion, and modification of previous results. Automated techniques for mapping out and exploring these complicated structures would be very helpful in moving beyond these limitations. This would not only aid human experts in space-mission planning. Automatic computation of reachability sets would also be useful in long-range space exploration, where the craft has a small window of opportunity in which to plan ahead—e.g., avoiding an impact—and Earth may be too far for fast communication. An onboard computer that could automatically calculate the reachabil-

ity set from the craft's current location, given its available ΔV budget, could make intelligent decisions autonomously in such situations.

The challenge here is to model the complex geometry of a 3D set as it evolves nonlinearly in 6D space, and to do so in a way that balances accuracy and efficiency. To accomplish this, we begin by choosing a set of state-space points in the volume of the ΔV sphere, then use those points to generate a mesh of 3D simplices that form an approximation of that volume. We then compute the forward trajectories of the points, monitoring the evolving geometry of the mesh and adjusting it based on the degree of state-space divergence that manifests via the deformation of the simplices. If a simplex grows quickly or becomes skewed, for instance, we add points in that region We then rebuild the mesh and iterate the process until a specified stopping condition (resolution, number of simplices, etc.) is reached. This stopping condition is one of several parameters that can be used to tune the algorithm along the accuracy-runtime spectrum. All of this has the desired effect of focusing the computational effort on the dynamically complex regions, which allows us to track the evolving reachability set effectively and efficiently.

Problem

The dynamical problem we choose to study is the Circular Restricted 3-Body Problem (CRTBP), which is a canonical model for chaotic space systems. The study of this problem, which dates back to Euler and Lagrange, led to the initial identification of chaotic and unstable systems by Poincaré. The CRTBP equations are a good model for satellite motions in the Earth-Moon system, the solar system, and about binary asteroids, among others. When fully normalized, the problem only has a single free parameter (μ), which describes how the mass is distributed between the two major bodies. See (Roy 2005) for more discussion of this problem.

The CRTBP consists of two bodies with mass that orbit each other in a circular orbit and a massless spacecraft that travels subject to the gravitational attraction of the major bodies The problem is posed in a rotating frame of reference, in which the two major bodies do not move. In this system, a spacecraft can have one of four fates, depending on its initial condition: it can orbit stably, it can escape, or it can impact either of the two major bodies. We represent the state of the craft using positions and velocities in three dimensions, model its dynamics using Newton's law of universal gravitation, and solve the associated differential equations using Mathematica's `NDSolve` method. All units—masses, gravitational constant, time, distance—are normalized, as is customary in problems like this, meaning that one solution can be scaled to different physical situations.

The reachability set for a single fuel burn at time t_0 begins life as a three-dimensional object—the ΔV sphere introduced in the previous section—embedded in the spacecraft's six-dimensional state space. The three dimensions of the sphere are the azimuth and elevation of the burn vector, and the magnitude (ΔV) of the impulse. Note that these are not the same as the spacecraft's velocity state variables. This is a hint at one of the subtle issues here: working with 6D geometry (let alone topology!) is very hard, but luckily

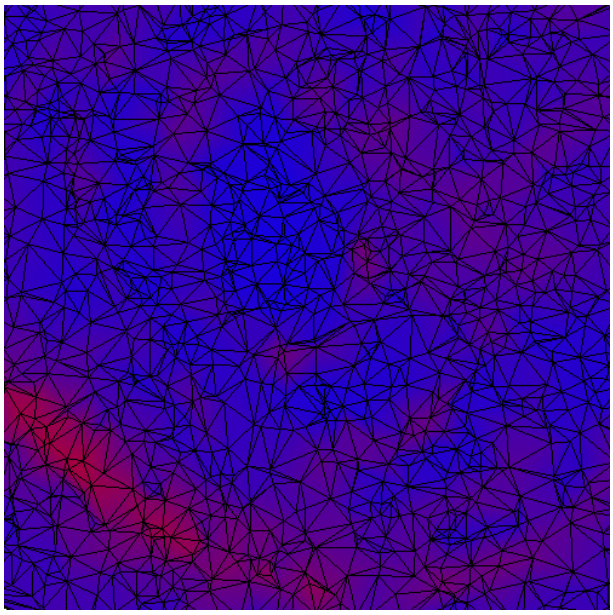

Figure 1: The geometry of a uniformly distributed random mesh on a ΔV disk. The black lines are the boundaries of the 2D simplices that tile this disk; the color scale represents the size of the corresponding 6D state-space volumes of each simplex Δt time units later, ranging from large (red) to small (blue). A grayscale version appears in (Komendera 2012).

the object that we are tracking is 3D, so we can extract those dimensions, work with them separately, and then perform a vector addition to move back to the full 6D state space. The first step in which this separation of dimensions becomes an issue is the initial model of the ΔV sphere. We fill the volume of this object with a uniform distribution of points, then use Delaunay triangulation (Watson 1981) to construct a mesh of non-overlapping tetrahedra—3D simplices—whose vertices are those points. We then evolve that volume forward in time by computing the forward trajectory from the 6D point that corresponds to each of those vertices, while keeping track of its topology via the relationships of those 3D simplices. This complex geometric information is obviously hard to visualize. Figure 1 shows a small patch of the surface of a ΔV disk with the simplex boundaries shown in black and the sizes of the corresponding 6D-embedded surfaces depicted with a color scale.

Done naïvely—working only with the original points and retaining all of the original edge relationships—this approach is suboptimal. Some trajectories with close starting conditions in the ΔV sphere (the red regions in Figure 1) diverge quickly, while others—the blue ones—diverge slowly. As time progresses, the fixed initial uniform distribution of vertices will thus come to overcharacterize the latter regions while not providing enough detail in the former. This can be seen in Figure 1, where there is no obvious relationship between the size of the simplices and the color scale

To avoid this, an intelligent reachability set tracking algorithm must be able to (a) distinguish between these regions

and (b) adapt the mesh accordingly. We have evaluated several heuristics for putting step (a) into practice, all of which trigger the addition of mesh points based on important aspects of simplex geometry: area, perimeter length, skew, and so on. Again, the dimensions of the problem make all of this more subtle that it initially appears; the heuristics work with the full 6D geometry, but the actions that they trigger affect the 3D ΔV sphere. There are other subtleties involved in the process of adding points to a reachability-set mesh: an algorithm that does so only along the boundaries of a simplex, for instance, will never explore its interior. To address this, we add points in a spatial distribution centered on the simplex center. This raises another important and subtle issue. If the evolving 'front' of the reachability set has an unseen fold that is straddled by the endpoints of two sample trajectories, then the associated simplex will straddle the boundary of the reachability set, and one cannot assume that all of its interior points are in that set. In a complex nonlinear system, where state-space volumes can be folded in arbitrarily complex ways, there is no effective way to work around this problem *post facto*. For this reason, our algorithm goes back to the original ΔV sphere before performing that operation: if a simplex on the Δt reachability set is chosen for division, we add the new points to the corresponding simplex on the original ΔV sphere, then compute their Δt forward trajectories. That is, our algorithm does not stop, adapt the mesh, and then move on; rather, it tracks the mesh out to the specified ending time (Δt), examines it to decide where points need to be added, then rewinds to the initial time, adds them, and evolves them forwards by Δt . It then loops until the stopping condition is reached.

The next section outlines the steps of this algorithm; the following one evaluates its results in the context of the CRTBP, using a heuristic that increases the mesh resolution when simplices are large and a point-placement strategy called the simplex shell method. As is implicit in the previous paragraphs, there are a number of free parameters in this algorithm: how the volume of a simplex affects resampling, the width of the distribution used to add points, and so on. These parameters, their roles in each step of the algorithm, and the process by which we chose their values are discussed briefly below; for more detail, please consult (Komendera 2012).

Algorithm

The algorithm for computing the Δt reachability set from an initial condition $\vec{x}_0(t_0)$, given a single impulse burn of magnitude $\in [0, \Delta V]$, has the following steps:

1. Create an initial mesh from points in the ΔV sphere
2. Compute the forward trajectory from each of those points
3. Remove any trajectories that blow up
4. Apply the chosen heuristic to the resulting mesh; sort the simplices according to the resulting values
5. Choose a set of simplices from that sorted list; add one point to the interior of each of the corresponding simplices in the initial ΔV sphere
6. Compute the forward trajectories from those new points, again removing any that blow up
7. Rebuild the mesh
8. Iterate steps 4–7 until the specified stopping condition is reached

Working in the 3D ΔV-sphere space, the algorithm seeds the initial mesh with a random uniform Euclidean distribution of points on and in that sphere, then uses Delaunay triangulation to fill the ΔV sphere with a set of simplices (the black lines in Figure 1) whose vertices are those points. The topology of this mesh is recorded and the 3D geometric information is mapped back to the full 6D state space via vector addition to $\vec{x}_0$. Next, a Δt -long trajectory from each of the resulting 6D points is generated using `NDSolve`.

The operative heuristic is then applied to the 6D geometric information associated with every simplex. Perhaps the simplest way to identify regions of the mesh where neighboring trajectories diverge quickly is to measure the size of each simplex—what we call the *simplex volume heuristic*. This is the calculation that produced the color scale values for Figure 1. Our implementation of this heuristic calculates the volume using the Cayley-Menger Determinant (Sommerville 1929); in order to avoid numerical issues, it skips simplices that are smaller than a specified minimum size. Simplex volume is not the only effective diagnostic, of course; we have also evaluated heuristics that measure perimeter growth, skew, and so on. Because of space limitations, we only discuss the volume heuristic in this paper. We also describe a control case, where no heuristic is applied and the evolving mesh is defined by the endpoints of the forward trajectories from the original points.

The algorithm then sorts the simplices according to the value returned by the heuristic, selects a specified number of simplices from this list—choosing those with higher weights more frequently—and places a new point within each of the corresponding simplices on the initial ΔV sphere. Since a given simplex may be chosen more than once during this process—indeed, that is part of the intent of the weighted sort—placing that new point at the exact center could create degeneracies. Instead, we use the *simplex shell method*, which randomly distributes vertices in a simplex using a variant of the normal distribution. First, a positive value is chosen from a normal distribution with a mean of 0 and a specified standard deviation σ. The σ parameter dictates the size of the simplex shell on whose surface the new vertex will appear. If $\sigma = 1$, for instance, the new point will appear inside that simplex 68.3% of the time; if $\sigma = 0.5$, that probability rises to 95.5%. This parameter allows one to control how close to the simplex center the added points will fall.

Finally, the added points are transformed to 6D state space and forward trajectories are computed from each one using the same solver and parameters, bringing them into temporal synchronization with the rest of the vertices on the Δt reachability set. The mesh that approximates that set is then rebuilt, incorporating the endpoints of these trajectories.

The steps described in the previous three paragraphs compose one "round" of the reachability set algorithm. Note that the adjustment to the mesh that is performed in a sin-

gle round not only improves the accuracy of the approximation to the reachability set, but may also expose new areas of concern. For this reason, we provide an outer loop to iterate multiple rounds. Note that this serves a very different function than simply selecting more points in a single round. Successive rounds build upon one another in a manner that iteratively refines the exploration.

Performance

We evaluated the performance of this reachability set algorithm using the CRTBP introduced earlier. The two major bodies had masses of 0.8 and 0.2 and were located on the x axis of the rotating frame, at $x = -0.2$ and $x = 0.8$. We explored reachability sets for spacecraft starting from three different initial conditions, all with $\Delta V = 2.5$, and varied the time horizon Δt from 1.0 to 5.0. The initial conditions were chosen to be representative of the different kinds of behavior that can occur in the CRTBP: one location between the two major bodies, where the dynamics are very complex, and two others (one outside the major bodies and one above them) where the behavior is somewhat simpler. The ΔV value was also chosen so as to tap into the complicated and interesting dynamics of the CRTBP. In all cases, we tested two versions of the algorithm: one with the simplex area heuristic and one with no simplex division (as a control case). In order to make the comparison fair, we fixed the number of vertices in the mesh of the final versions of all of the resulting reachability sets to be 5000.

We defined the error of an approximation to a Δt reachability set as the average of the Euclidean distance in state space between the predicted location of its trajectories (calculated using interpolation on the reachability set mesh) and the actual location of those trajectories (calculated using `NDSolve`). To estimate this, we used a Monte Carlo approach, placing a number of uniformly distributed random points on and in the ΔV sphere, determining which simplices contain those points, and computing their forward trajectories. We used barycentric coordinates to interpolate each point's predicted state in a simplex at a time Δt , calculated the 6D Euclidean distance between the point's predicted state and its actual state, and then averaged across the points. For each run of the reachability set algorithm, we repeated this Monte Carlo error calculation five times and averaged the results. We repeated the entire process over three runs, for a total of 15 error measurements over three different calculations of each reachability set.

Our first series of tests involved a common situation in space mission planning: a planar orbit. In this situation, the ΔV sphere is actually a *disk*, since the spacecraft only moves (and thrusts) in the orbital plane of the two major bodies. The image in Figure 1 came from this example, with 5000 points distributed on a ΔV disk in the $x - y$ plane, centered at $x_0 = 0.5$—i.e., a spacecraft positioned directly between the two major bodies, with thrust allowed only in the orbital plane. When the reachability set at $\Delta t = 5.0$ is calculated from this initial mesh without any subdivision, the average error is 0.46 in normalized units. When the simplex volume heuristic is used to tailor the mesh to the evolving dynamics, that error falls to 0.39—a 14.9% improvement without any increase in the size of the computational mesh. The effects of the heuristic-driven subdivision are clearly visible in Figure 2, where—in contrast to the fixed mesh of Figure 1—the red regions are covered by much finer simplices. This produces a better approximation to the true reachability set and thereby reduces the error. The reachability set shown in Figure 2 was computed using an initial mesh of 1000 vertices, with 80 rounds of 50 vertex additions and a simplex factor $\sigma = 1$. If we add the same number of vertices, but in more rounds, the error is slightly better. This is the power of iterative improvement. If we decrease σ, thereby biasing the positioning of new points towards the center of the simplex that is being subdivided, the error steadily improves. This is because smaller σs force a more-even division of space. Beyond a certain point, however, this improvement ceases, as tiny σ can create degeneracy. Tiny *simplices* can also cause problems, but for a different reason: without a lower bound on the minimum simplex size, the subdivision could overemphasize some regions at the expense of the rest of the mesh. This bound must be chosen carefully, however: setting it too high will spread out the mesh and negate the effects of the subdivision. An appropriate way to do this is to fix all other variables and find the bound that minimizes the error. In Figure 2, for instance, this lower bound was set to 0.0001 square units; lowering it to 0.000001 increased the error back to (and beyond) that of the fixed mesh.

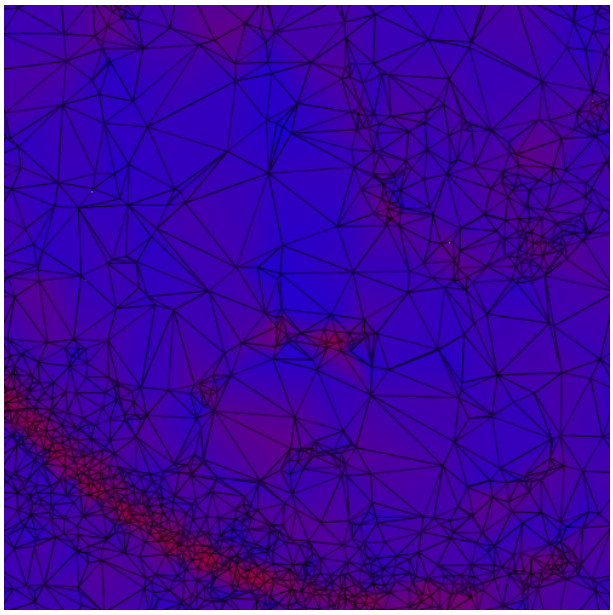

Figure 2: The same setup as Figure 1, but with the simplex volume heuristic used to focus the resolution of the reachability set mesh. The color scale, as before, represents the volume of the 6D state-space volumes corresponding to each simplex. The success of the algorithm is evident from the fine distribution of the simplices in the redder regions, where the system trajectories diverge, and the comparatively coarser mesh in the blue regions, where the dynamics are less complicated and the reachability set is smoother.

In dynamically simpler regions, the errors were uniformly better: 0.254 and 0.339 for the fixed-mesh $\Delta t = 5.0$ reachability sets starting at $(x_0, y_0) = (1.3, 0)$ and $(x_0, y_0) = (0, 1.0)$, respectively—compared to 0.460 for the $(x_0, y_0) = (0.5, 0)$ case described in the previous paragraph. The simplex volume heuristic also got more traction in runs from these initial conditions, yielding 38.2% and 60.0% improvements over the corresponding fixed-mesh error values, respectively, with $\sigma = 0.1$. The reason for this is that the curvature of the spacecraft's trajectories is smaller in both of these regions—and even more so in the latter—which plays directly to the strengths of the heuristic.

In a second series of tests, we relaxed the planar restriction and worked with a true ΔVsphere from the same initial conditions. The errors here were generally larger than in the disk case: 0.556 in the $(x_0, y_0, z_0) = (0.5, 0, 0)$ case, for instance, versus the 0.460 average error for the ΔVdisk computed from the same starting point with the same parameter values. This makes complete sense because we are using the same fixed number of vertices (5000) to cover a higher-dimensional object. Subdividing the simplices yielded mixed results in the spherical-thrust case. In the dynamically complex region, it was only slightly advantageous and sometimes even made the error slightly worse. Matters were better for the reachability sets from the other two initial conditions, where the errors for the fixed mesh cases were actually *better* than the corresponding planar cases (0.246 and 0.240) and subdivision was helpful in both situations (15.6% and 15.9% improvement over the fixed-mesh control case, respectively). As in the planar case, smaller σ values generally improved the results of the subdivision, and setting the lower bound for simplex division at too low a value negated all of that process's good effects.

For all initial conditions, parameter values, and geometries, the average error was smaller at $\Delta t = 1.0$ than $\Delta t = 5.0$. That too makes sense, since the complexity of a reachability set grows over time.

The results of these experiments, which are tabulated in Table 1, indicate that this reachability set calculation strategy appears to work quite well. Adaptively adding vertices to simplices that grow allows the algorithm to focus the computational effort in an intelligent fashion, producing meshes that were more accurate than static meshes with the same number of vertices. The algorithm's free parameters, which are designed to allow a user to balance accuracy against computational effort, are straightforward and easy to tune. The closer new vertices are to the centers of their parent simplices, for instance, the better the result. This improvement is more pronounced over shorter time spans, before the chaotic nature of the trajectories overwhelms the heuristic. Limiting the number of subdivisions that can occur in a small region of the ΔVsphere also has a profound effect on the result, as it allows the algorithm to focus its effort elsewhere after a certain region has been adequately explored.

2D ΔVdisk						
Δt	IC	R	V	σ	Min	Improvement
5.0	$(0.5, 0)$	80	50	1.0	10^{-4}	$14.9 \pm 5.5\%$
5.0	$(0.5, 0)$	800	5	1.0	10^{-4}	$15.5 \pm 3.6\%$
5.0	$(0.5, 0)$	800	5	0.5	10^{-4}	$16.9 \pm 6.5\%$
5.0	$(0.5, 0)$	800	5	0.2	10^{-4}	$22.1 \pm 7.1\%$
5.0	$(0.5, 0)$	800	5	0.1	10^{-4}	$24.2 \pm 4.0\%$
5.0	$(0.5, 0)$	800	5	1.0	10^{-6}	$-6.3 \pm 6.3\%$
5.0	$(1.3, 0)$	800	5	0.1	10^{-4}	$38.2 \pm 5.6\%$
5.0	$(0, 0.5)$	800	5	0.1	10^{-4}	$60.1 \pm 6.3\%$
1.0	$(0.5, 0)$	800	5	0.1	10^{-4}	$60.1 \pm 9.1\%$
3D ΔVsphere						
5.0	$(0.5, 0)$	800	5	1.0	10^{-4}	$1.3 \pm 5.4\%$
5.0	$(0.5, 0)$	800	5	0.2	10^{-4}	$3.4 \pm 5.3\%$
5.0	$(0.5, 0)$	800	5	0.1	10^{-4}	$4.9 \pm 5.2\%$
5.0	$(0.5, 0)$	800	5	1.0	10^{-6}	$-6.9 \pm 4.6\%$
5.0	$(1.3, 0)$	800	5	0.1	10^{-4}	$15.6 \pm 8.9\%$
5.0	$(0, 0.5)$	800	5	0.1	10^{-4}	$15.9 \pm 5.0\%$
1.0	$(0.5, 0)$	800	5	0.1	10^{-4}	$28.7 \pm 8.0\%$

Table 1: Table of errors for the simplex volume heuristic for different initial conditions (IC), numbers of rounds (R) and vertices added per round (V), simplex shell factor (σ), and minimum simplex size (Min). All errors are expressed as % improvement over the error of the corresponding fixed-mesh control case.

Future Work

The reachability set mapping algorithm presented here will be ready for deployment when it can run fast enough, on deployable hardware, to handle the small margins for error that this problem poses. The solutions in this paper represent significant progress towards that goal, but a variety of issues remain. The heuristic-driven subdivision is better than a fixed mesh, but there are several obvious avenues for further improvement. One could, for instance, apply a "flatness" measure to the simplex in order to measure its distortion over time. Also, the algorithm presented here never *deletes* vertices from the mesh, which would make sense in dynamically simple regions. In a situation with limited computational power—viz., a fixed number of vertices—this could further shift the effort to the areas where it is warranted.

It would also make sense to automatically identify impact and escape scenarios—regions that lead to mission failure—in reachability sets, and perhaps increase the resolution on their boundaries. This would not only map out a particularly critical set of regions; again, it would focus the mesh geometry in a highly appropriate way. And since there would be no need to explore the state-space volume inside a zone of impact scenarios, one could shift the associated computational effort to regions that do not lead to mission failure. A heuristic that combined some measure of simplex geometry (to measure dynamical complexity) and an assessment of impact/escape fate would work very well, since trajectories that pass close to a center of gravity tend to be highly curved. For example, in Figure 2, the long and narrow region at the bottom of the image would be considered an impact if that body had a radius of 0.1. Working alone, the simplex volume heuristic will force a fine-grained mesh refinement in these regions; if it were combined with an 'end result' heuristic that identified near-impacts, that (largely wasted)

effort could be saved.

Single impulse burns are only one kind of thrust. Multiple ΔV burns—a fixture of space missions—are another logical step, but they add to the already complex geometry of this problem (viz., seven new variables for each burn). The high-dimensional meshes in problems like this will require many more trajectories to properly characterize, but they represent a far richer set of maneuvers. Finally, the burns performed by low-thrust, high-specific-impulse ion thrusters, which are becoming increasingly common, cannot be treated as instantaneous ΔV changes. It would be ideal to compute reachability sets automatically for spacecraft with this capability, but continuous burns make this an infinite-dimensional planning problem. An effective approximate solution to that would be a real challenge, but of potentially enormous value to space mission designers.

Related Work

The field of artificial intelligence (AI) has grappled with spatial reasoning problems for years, but none of the resulting solutions has been applied to space mission design. AI techniques have been used to plan the *operations* of spacecraft—e.g., the Casper planner on the 3CS and ACS missions (Knight et al. 2001)—but have not been used to plan the paths of the craft through space. An efficient and automatic approach to this important envisioning problem could revolutionize the practice of orbit design, with additional implications for space situational awareness (Holzinger and Scheeres 2010). There has been some recent progress in this area; (Coffee, Anderson, and Lo 2011) introduced a computational method for extracting and using dynamical "channels" through the state space of the CRTBP, and adaptive space partitioning algorithms are frequently used for real-time collision detection (Ericson 2005).

While the three-body problem has been the subject of extensive research, techniques for representing a finite, evolving state-space volume in gravitational systems have not been explored as completely. In the aerospace literature, techniques for this involve either higher-order expansions about the nominal trajectory (Park and Scheeres 2006; Guibout and Scheeres 2006), multiple nonlinear propagations of a distribution of state-space points, or some hybrid of the two approaches (Fujimoto, Scheeres, and Alfriend 2011). The AI literature includes some work on the representation of evolving state-space volumes—notably the flow pipes of Zhao (Zhao 1992) and the more-general Spatial Aggregation Language that grew out of that work (Bailey-Kellogg, Zhao, and Yip 1994), as well as the work of Nishida and collaborators (Nishida et al. 1991)—but very little of that is ongoing.

Conclusion

Space mission design involves complex, high-dimensional spatial reasoning. Currently, experts approach that task by working incrementally from previously identified solutions, which greatly limits their exploration of the design space. The goal of the work described here is to map out solutions to this complex, high-dimensional problem automatically. The key to our solutions is a careful treatment of time, space, and sensitivity. Our algorithm tracks evolving reachability sets effectively and efficiently by relying on a heuristic to identify regions where neighboring trajectories diverge quickly, and then focusing the computational mesh accordingly. The results are promising: for the same number of vertices, the mesh produced by this intelligent subdivision approach was a significantly better approximation to the true geometry of the reachability set. This could not only aid human mission planners, but also potentially enable autonomous onboard mission planning for spacecraft whose communication with Earth is limited.

References

Bailey-Kellogg, C.; Zhao, F.; and Yip, K. 1994. Spatial aggregation: Language and applications. In *Proceedings AAAI-94*, 517–522.

Coffee, T.; Anderson, R.; and Lo, M. 2011. Multiobjective optimization of low-energy trajectories using optimal control on dynamical channels. In *AAS/AIAA Space Flight Mechanics Meeting*.

Ericson C. 2005. *Real-Time Collision Detection*. Morgan-Kaufman, San Francisco, CA.

Fujimoto, K.; Scheeres, D.; and Alfriend, K. 2011. Analytical non-linear propagation of uncertainty in the two-body problem. In *AAS/AIAA Spaceflight Mechanics Meeting*.

Guibout, V., and Scheeres, D. 2006. Spacecraft formation dynamics and design. *Journal of Guidance, Control, and Dynamics* 29(1):121–133.

Holzinger, M., and Scheeres, D. 2010. Object correlation, maneuver detection, and maneuver characterization using control effort metrics with uncertain boundary conditions and measurements. In *Proceedings of the 2010 AIAA GNC Conference*.

Knight, S.; Rabideau, G.; Chien, S.; Engelhardt, B.; and Sherwood, R. 2001. Casper: space exploration through continuous planning. *Intelligent Systems, IEEE* 16(5):70 – 75.

Komendera, E. 2012. Description of the reachability set adaptive mesh algorithm. Technical Report CU-CS-1090-12, University of Colorado at Boulder.

Nishida, T.; Mizutani, K.; Kubota, A.; and Doshita, S. 1991. Automated phase portrait analysis by integrating qualitative and quantitative analysis. In *Proceedings of the ninth National conference on Artificial intelligence*, volume 2.

Park, R., and Scheeres, D. 2006. Nonlinear mapping of gaussian state uncertainties: Theory and applications to spacecraft control and navigation. *Journal of Guidance, Control, and Dynamics* 29(6):1367–1375.

Roy, A. 2005. *Orbital motion, 4th Ed.* Institute of Physics.

Sommerville, D. M. Y. 1929. *An Introduction to the Geometry of n Dimensions*. New York: Dover.

Watson, D. F. 1981. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal* 24(2):167–172.

Zhao, F. 1992. Automatic analysis and synthesis of controllers for dynamical systems based on phase-space knowledge. *PhD Thesis*.