Nonlinear and Hybrid Control Via RRTs

Michael S. Branicky and Michael M. Curtiss
Electrical Engineering and Computer Science Dept.
Case Western Reserve University
`{msb11,mmc18}po.cwru.edu`

Abstract

In this paper, we review rapidly-exploring random trees (RRTs) for motion planning, experiment with them on standard control problems, and extend them to the case of hybrid systems.

1 Introduction and Overview

Researchers in the computer science and control theory communities have produced many models for describing the dynamics of hybrid systems (e.g., see [3, 25, 12, 26]). For the purpose of the discussion in this document, we consider a simple illustrative case, in which the constituent continuous state and input spaces (in each mode) are the same. Thus, we have a hybrid system of the form

$$\begin{array}{rcll} \dot{x} & = & f(x,u,q), & x \notin J(x,u,q) \\ (x,q)^+ & = & D(x,u,q), & x \in J(x,u,q). \end{array} \tag{1.1}$$

Here, $x \in X$ is the continuous state, $u \in U$ is the input, and $q \in Q \simeq \{1, 2, \ldots, N\}$ is the discrete state or *mode*. Also, $f(\cdot,\cdot,q)$ is the continuous dynamics, $J(\cdot,\cdot,q)$ is the jump set, and $D(\cdot,\cdot,q)$ is the discrete transition map, all for mode q. The map D relates the post-jump hybrid state $(x,q)^+$ from the pre-jump hybrid state (x,q). The input u, which can include both continuous and discrete components, allows the introduction of non-determinism in the model, and can be used to represent the action of control algorithms and the effect of environmental disturbances. The evolution of the discrete state q models switches in the control laws and discrete events in the environment, such as failures.

Briefly, the dynamics are as follows: the system starts at hybrid state $(x(t_0), q_0)$ and evolves according to $f(\cdot,\cdot,q_0)$, until the set $J(\cdot,\cdot,q_0)$ is reached. At this time, say t_1, the continuous and/or discrete state instantaneously jump to the hybrid state $(x(t_1^+), q_1) = D(x(t_1), u, q_0)$, from which the evolution continues. While terse, the above model encompasses both autonomous and controlled switching and jumps, and allows modeling of a large class of embedded systems, including for example ground, air and space vehicles and robots; see [4, 3] for more details.

Given a model of the system in the form (1.1), the path planning problem can be shortly stated as finding a (controlled) trajectory of the system that leads from a start to a goal configuration.

Attempts to fight the curse of dimensionality have led to the introduction of randomized (or Monte-Carlo) approaches to path planning that are capable of solving many challenging problems efficiently, at the expense of being able to guarantee that a solution will be found in finite time. Most randomized planning methods are designed for the generalized mover's problem, including randomized potential fields [2, 10, 19], probabilistic roadmaps [1, 18], Ariande's clew algorithm [28, 27], and the planners in [17, 33]. Derandomization of some of these algorithms has been explored in [6, 7].

Many path planning methods, including dynamic programming and most of the randomized planning methods can be categorized as incremental search methods. In these methods a tree (or two trees in the bidirectional version) is grown incrementally from the initial state by adding a new edge and vertex in each iteration after performing some local motion. There are generally two decision problems at each iteration: 1) which vertex should be selected for expansion? 2) what local motion should be executed? The answers to these questions are given by the particular method. For example, dynamic programming selects the "active" vertex with the lowest cost, and the local motion attempts to move in a direction not yet considered. Among path planning techniques, incremental search methods are most amenable to the inclusion of differential constraints.

An incremental search method that has achieved considerable success in the design of feasible trajectories is the Rapidly-exploring Random Tree (RRT) [20, 21]. This method can solve challenging problems that involve state spaces of up to twelve dimensions with the inclusion of both differential constraints and complicated obstacle constraints. The extensions of these ideas to hybrid systems has not yet been considered in the literature.

Other approaches to hybrid control problems are presented in [8, 9, 5, 32].

2 RRT Background

We describe our control approach in terms of the Rapidly-exploring Random Tree (RRT), which was introduced in [20, 21] as an exploration algorithm for quickly searching high-dimensional spaces that have both global constraints (arising from workspace obstacles and velocity bounds) and differential constraints (arising from kinematics and dynamics). The key idea is to bias the exploration toward unexplored portions of the space by randomly sampling points in the state space, and incrementally "pulling" the search tree toward them. The resulting method is much more efficient than brute-force exploration of the state space. The description below serves as a basis to illustrate the main ideas.

The basic RRT construction algorithm is given in Figure 1 (left). A simple iteration is performed in which each step attempts to extend the RRT by adding a new vertex that is biased by a randomly-selected state, $x \in X$. The EXTEND function selects the nearest vertex already in the RRT to x. The "nearest" vertex is chosen according to the metric, ρ. The function NEW_STATE makes a motion toward x by applying an input $u \in U$ for some time increment Δt. This input can be chosen at random, or selected by trying all

```
BUILD_RRT($x_{init}$)
1  $T$.init($x_{init}$);
2  for $k = 1$ to $K$ do
3      $x_{rand} \leftarrow$ RANDOM_STATE();
4      EXTEND($T, x_{rand}$);
5  Return $T$
```

```
EXTEND($T, x$)
1  $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x, T$);
2  if NEW_STATE($x, x_{near}, x_{new}, u_{new}$) then
3      $T$.add_vertex($x_{new}$);
4      $T$.add_edge($x_{near}, x_{new}, u_{new}$);
```

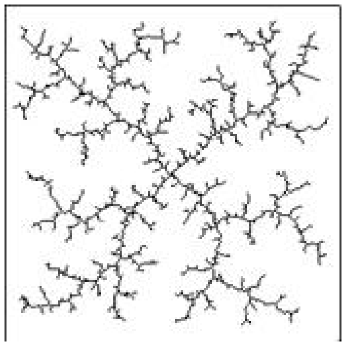

Figure 1: The basic RRT construction algorithm (left) and an example RRT (right)

possible inputs and choosing the one that yields a new state as close as possible to the sample, x (if U is infinite, then a finite approximation or analytical technique can be used). NEW_STATE implicitly uses the collision detection function to determine whether the new state (and all intermediate states) satisfy the global constraints. For many problems, this can be performed quickly ("almost constant time") using incremental distance computation algorithms [16, 24, 29] by storing the relevant invariants with each of the RRT vertices. If NEW_STATE is successful, the new state and input are represented in x_{new} and u_{new}, respectively. The left column of Figure 1 shows an RRT grown from the center of a square region in the plane. In this example, there are no differential constraints (motion in any direction is possible from any point). The incremental construction method biases the RRT to rapidly explore in the beginning, and then converge to a uniform coverage of the space [22]. The exploration is naturally biased towards vertices that have larger Voronoi regions. This causes the exploration to occur mostly on the unexplored portion of the state space.

In addition to growing a tree from the starting state, many RRT implementations grow a second tree from the goal state. Such trees grow in four steps:

1. Grow start-tree towards a random unexplored configuration.
2. Grow goal-tree towards a random unexplored configuration.
3. Grow start tree towards goal tree. At each iteration, select a random node in the goal tree to grow towards it.
4. Grow goal tree towards start tree. A solution path is found when the two trees finally connect.

3 RRTs for Nonlinear Control

Other researchers have applied RRT's to planning problems of various types including path-steering, manipulation planning for digital actors, varieties of holonomic planning, and attitude control (kinodynamic planning) [21, 23, 11]. To our knowledge, we are the first experimenters to test RRT's on standard control problems. It is our hope that by studying the RRT's performance in these common problems, we will be able to gauge the strengths and weaknesses of RRT's compared to other approaches.

3.1 Pendulum Swing-Up

The first experiment we conducted was applying the RRT to the swing-up problem for a nonlinear pendulum:

Definition 3.1 *Pendulum Swing-Up Given:*

- *a pendulum of mass m and length l with equation of motion*

$$\ddot{\theta} = \frac{-3g}{2l} \sin\theta - \frac{3\tau}{ml^2}$$

- *Motor at tip which can apply torques of $\tau \in \{-1, 0, 1\}$ units*
- *Initial state of $\theta = 0$ (down) and $\dot{\theta} = 0$*
- *Goal state of $\theta = \pi$ (up) and $\dot{\theta} = 0$*

The goal for the planner is to find a series of torque-time pairs that get the pendulum to the goal state. In all but the most trivial cases, the motor is unable to lift the pendulum to the goal state in one smooth motion. The pendulum therefore must be swung back and forth until it achieves sufficient velocity to reach the goal configuration. Our first try at solving the problem, a single-tree RRT using the straightforward Euclidean metric, $\rho = \sqrt{(\Delta\theta)^2 + (\Delta\dot{\theta})^2}$, proved to be quite successful. Usually finding a solution in less than 10,000 iterations (only a few seconds of computation on most modern computers), our implementation showed that the RRT algorithm is both fast and adaptable to many problem domains. See Figure 2 (left).

The dual-tree solution to the same problem was also impressive, sometimes finding a path to the goal state in close to half the time of its single-tree relative. One interesting characteristic of the solution trees is how clearly it demonstrates the dynamics of the system. See Figure 2 (right).

3.2 Acrobot

For our second experiment, we tested the RRT algorithm on a problem of higher dimensionality. The acrobot has gained attention in recent literature as an interesting control task in the area of reinforcement learning [9]. Analogous to a gymnast swinging on a high-bar, the acrobot has been studied by both control engineers and machine learning researchers. The equations of motion used come from [31, p. 271]. A time step of 0.05 seconds was used in the simulation, with actions chosen after every four time steps. The torque applied at the second joint is denoted by $\tau \in \{-1, 0, 1\}$. There were no constraints on the joint positions, but the

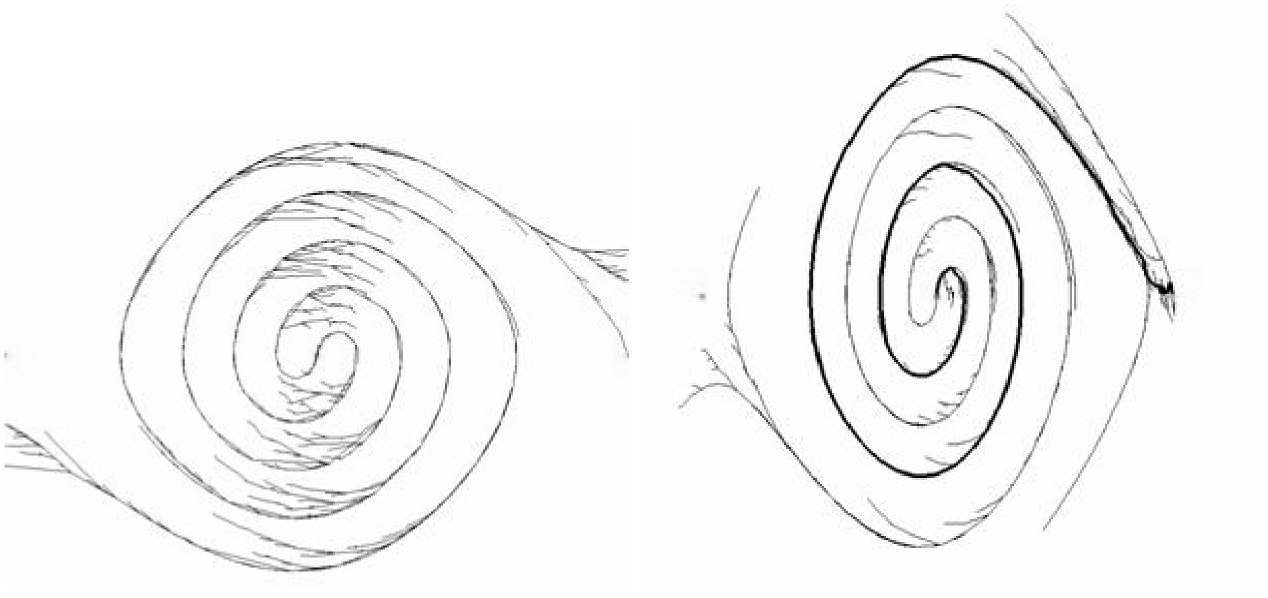

Figure 2: Single- and Dual-RRT Solutions to the Pendulum Swing-Up Problem. The x-axis corresponds to θ and the x-axis to $\dot{\theta}$. The left image shows a single-tree RRT solution for the pendulum problem after 5600 iterations. The right image shows a dual-tree RRT solution after 3300 iterations (solution in dark).

angular velocities were limited to $\dot{\theta}_1 \in [-4\pi, 4\pi]$ and $\dot{\theta}_2 \in [-9\pi, 9\pi]$. The constants were $m_1 = m_2 = 1$ (masses of the links), $l1 = l2 = 1$ (lengths of links), $l_{c1} = l_{c2} = .5$ (lengths to center of mass of links), $I_1 = I_2 = 1$ (moments of inertia of links), and $g = 9.8$ (gravitational constant).

There are numerous goals that planning systems can attempt to reach when controlling the acrobot, but most involve reaching various vertical levels. In our testing, we attempted to swing the tip of the acrobot above some vertical level, $y = y_{goal}$. The single-tree RRT had no problem finding a solution to the acrobot tip-goal problem. One question that we may investigate further is how well the RRT-based solution compares quantitatively to other planners. Unlike some of the competing planners, the RRT is based on virtually no domain-specific knowledge except for the acrobot's equations of motion, yet the RRT planner was able to perform well compared to published metrics of energy efficiency and time efficiency [31].

Figure 3 shows the vertical position of the tip of one version of the RRT-controlled acrobot versus time. Like the inverted pendulum, the acrobot had to swing back and forth multiple times in order to reach the goal state. The starting position was $y = -2.0$ and $y_{goal} = 1.0$. As for the time-lapse behavior, the RRT-controlled acrobot showed similar behavior to that shown in [31, p. 274].

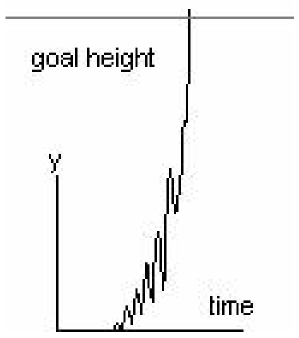

Figure 3: Acrobot Swing-Up Problem: vertical position versus time

4 RRTs for Hybrid Systems

Emilio Frazzoli and his co-workers have used random search in the context of a hybrid "manuever automaton" to plan motions for aerospace vehicles [14, 5, 13, 15]. However, we believe our work is the first general description of a hybrid RRT.

A general, hybrid RRT can be achieved in various ways, depending on the underlying hybrid systems model and specifics of the continuous and discrete dynamics (and symmetries therein). We now wish to give a taste of the way a hybrid RRT might work for the model (1.1). A planning/control problem will have a *target set* $T \subset X \times Q$.

The simplest algorithm one might envision would explore reachable space by growing a *forest* of RRTs, one in each mode, with jump points among various trees in the forest identified. In the more general case, evolution will start from a set of seeds in a *start set* $S \subset X \times Q$, encompassing one or more modes, and proceed from there according the algorithm outlined below. One may think of the resulting tree as (a) growing in the hybrid state space, $X \times Q$, or (b) as growing in X, with nodes and arcs colored/labeled by the current mode.

Even under this setup, there are several cases to consider:

1. General specifications; S, T, J, and D are arbitrary.

2. Homogeneous specifications: $S = B \times Q$ and $T = G \times Q$. i.e., the start and target sets are independent of mode.

3. Homogeneous switching: $J(x, q) \equiv J(x)$ and $D(x, q) \equiv D(x)$, independent of q.

4. Unrestricted switching: $J(\cdot, q) = X$ for all q and $D(x, q) = x$ for all x, q.

While the above is not exhaustive, it provides a sense of a few types of symmetries in the discrete dynamics that can be exploited by the algorithm.

In the case of unrestricted switching, the hybrid RRT algorithm is exactly the same as outlined above, except that the control set is augmented to allow mode changes: $U \mapsto U \times Q$. The other cases are non-trivial. In the case of homogeneous specifications, x_{rand} lives, and distances are measured in, the continuous state space X; in the general case, x_{rand} lives, and distances are measured in, the hybrid state space $X \times Q$. The latter brings up the issue of designing metrics for combined continuous and discrete space, which is a topic of current research. In either case, the NEW-STATE function must respect the hybrid dynamics. Typically, for purely continuous RRTs, the states examined come from extending the state x_{near} according to the dynamics $f(x, \cdot)$ for a fixed time and for various (sampled) $u \in U$. In the hybrid case, this continues to hold for (x_{near}, q_{near}) if there are no intersections with the jump set $J(\cdot, q_{near})$. If there are, evolution continues from the destination point(s), using the same or different u, until the desired amount of time elapses.

In Figure 4 we give an example of a hybrid RRT. Pictured from left to right in each row are four square floors, 1 through 4. Stairs (jumps) are given by triangles, with destinations given by inverted triangles in the next highest floor. The tree started in the gray square in the center of floor 1, and the target set is the gray square on floor 4. Successive rows represent different stages in the expansion process. The hybrid state is $s = (x, y, q) \in [-20, 20] \times [-20, 20] \times \{1, 2, 3, 4\}$. The distance metric used is $\rho(s_1, s_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} + 20|q_1 - q_2|$.

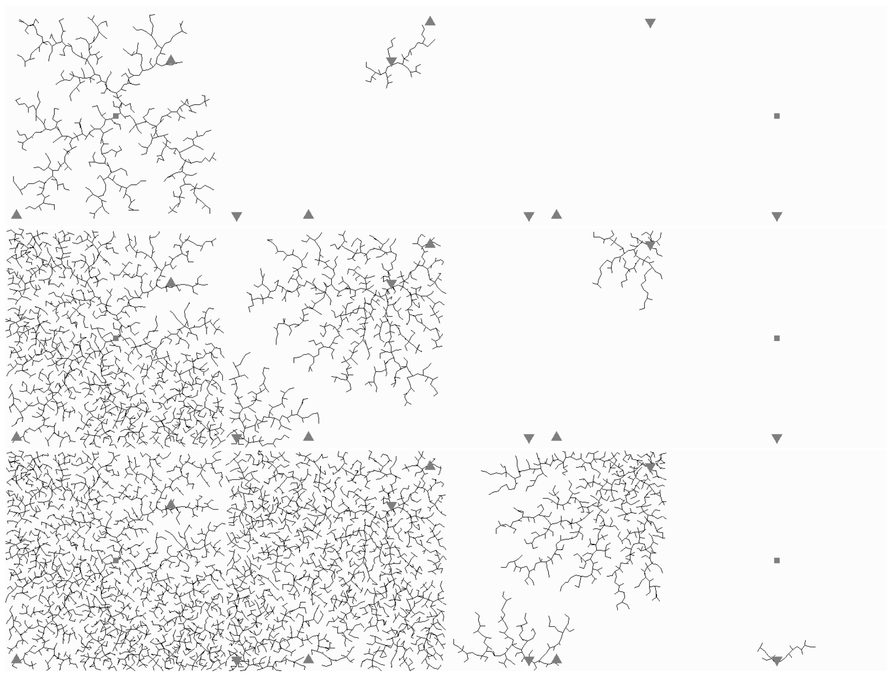

Figure 4: Stair Climbing: an example hybrid RRT.

References

[1] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *IEEE Int. Conf. Robot. & Autom.*, pages 113–120, 1996.

[2] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10(6):628–649, December 1991.

[3] M. S. Branicky. *Studies in Hybrid Systems: Modeling, Analysis, and Control.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[4] M.S. Branicky, V.S. Borkar, and S.K. Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE Trans. Automatic Control*, **43**(1):31–45, 1998.

[5] M.S. Branicky, T.A. Johansen, I. Petersen, and E. Frazzoli. On-line techniques for behavioral programming. *Proc. IEEE Conf. on Decision and Control*, Sydney, AUSTRALIA, December 2000.

[6] M.S. Branicky, S.M. LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. *Proc. IEEE International Conf. Robotics and Automation*, pp. 1481–1487, Seoul, KOREA, May 2001.

[7] Michael S. Branicky, Steven M. LaValle, Kari Olson, and Libo Yang. Deterministic vs. Probabilistic Roadmaps. *IEEE Trans. on Robotics and Automation*, Jan. 2002. Submitted. Electronically available at `http://dora.cwru.edu/msb/pubs/BLOY2002.ps`

[8] M.S. Branicky and S.K. Mitter. Algorithms for optimal hybrid control. *Proc. IEEE Conf. Decision and Control*, New Orleans, LA, pp. 2661–2666, Dec. 1995.

[9] M.S. Branicky and G. Zhang. Solving hybrid control problems: Level sets and behavioral programming. *Proc. American Control Conf.*, Chicago, IL, June 28–30, 2000.

[10] D. Challou, D. Boley, M. Gini, and V. Kumar. A parallel formulation of informed randomized search for robot motion planning problems. In *IEEE Int. Conf. Robot. & Autom.*, pages 709–714, 1995.

[11] Peng Cheng and Zuojun Shen and Steven M. LaValle. Using Randomization to Find and Optimize Feasible Trajectories for Nonlinear Systems, by *Proc. 38th Annual Allerton Conference on Communication, Control, and Computing.* 2000.

[12] J. M. Davoren and A. Nerode. Logics for hybrid systems. *Proceedings of the IEEE*, 88:985–1010, July 2000.

[13] E. Frazzoli. *Robust Hybrid Control for Autonomous Vehicle Motion Planning.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2001.

[14] E. Frazzoli, M.A. Dahleh, and E. Feron. A hybrid control architecture for aggressive maneuvering of autonomous helicopters. In *IEEE Conf. on Decision and Control*, December 1999.

[15] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control, and Dynamics*, May 2002. To appear.

[16] L. J. Guibas, D. Hsu, and L. Zhang. H-Walk: Hierarchical distance computation for moving convex bodies. In *Proc. ACM Symposium on Computational Geometry*, pages 265–273, 1999.

[17] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. J. Comput. Geom. & Appl.*, 4:495–512, 1999.

[18] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. & Autom.*, 12(4):566–580, June 1996.

[19] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.*, 5(1):90–98, 1986.

[20] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University, Oct. 1998.

[21] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 473–479, 1999.

[22] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Workshop on the Algorithmic Foundations of Robotics*, 2000.

[23] Steven M. LaValle and James J. Kuffner, Jr. Randomized Kinodynamic Planning. *International Journal of Robotics Research.* 20(5):378–400, May 2001.

[24] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Int. Conf. Robot. & Autom.*, 1991.

[25] J. Lygeros. *Hierarchical Hybrid Control of Large Scale systems.* PhD thesis, University of California, Berkeley, CA, 1996.

[26] N. Lynch, R. Segala, and F. Vandraager. Hybrid I/O automata revisited. In M.D. Di Benedetto and A. Sangiovanni-Vincentelli, editors, *Hybrid Systems IV: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 403–417. Springer-Verlag, 2001.

[27] E. Mazer, J. M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *J. Artificial Intell. Res.*, 9:295–316, November 1998.

[28] E. Mazer, G. Talbi, J. M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. In *Proc. Int. Conf. of Society of Adaptive Behavior*, Honolulu, 1992.

[29] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, Mitsubishi Electronics Research Laboratory, 1997.

[30] M.W. Spong. Swing up control of the acrobot. *Proc. IEEE Internationl Conf. on Robotics and Automation*, San Diego, CA, pp. 2356–2361, May 8–13, 1994.

[31] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[32] C. Tomlin, J. Lygeros, and S. Sastry, Computing controllers for nonlinear systems, *Proc. Hybrid Systems: Computation and Control*, Nijimegen, The Netherlands, March 1999.

[33] Y. Yu and K. Gupta. On sensor-based roadmap: A framework for motion planning for a manipulator arm in unknown environments. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 1919–1924, 1998.