Proceedings of the American Control Conference
Chicago, Illinois • June 2000

Solving Hybrid Control Problems: Level Sets and Behavioral Programming[1]

Michael S. Branicky and Gang Zhang
Electrical Engineering and Computer Science Department
Case Western Reserve University

Abstract

Hybrid systems include both continuous dynamics and discrete events. Here, we represent the continuous dynamics by differential equations and represent the events by a discrete transition model. We describe computational approaches to solving optimal hybrid control problems using two techniques: a fast marching level set method and behavioral programming. We review our extension of the fast marching level set method to the hybrid setting, including its formalization, a constructive proof of its correctness, approximation errors to the analog solution, and upper- and lower-bounding approximate solutions. Our work also explores an idea known as behavioral programming. We review the theoretical underpinnings and then perform some experiments using this technique to solve a specific problem in robotic assembly, the peg-in-hole problem. We demonstrate the abstraction of primitive actions into behaviors, try out several strategies for combining behaviors, and compare their optimality and computational effort vis-a-vis primitive actions.

1 Introduction

There are many hybrid system control problems around us, including robotic assembly tasks, automotive powertrains, flight control, power-switched electronics, etc. In general, they are those combining continuous and discrete states, inputs, and outputs along with coupled continuous and discrete dynamics, e.g., differential equations coupled with finite automata. In control theory and computer science, a mathematics of "hybrid systems" is emerging that enables the careful study of questions of analysis, control, and planning for this important realm of engineering control problems.

In [1, 4, 2, 7], Branicky and his co-workers formalized the modeling, analysis, and control of hybrid systems. In the case of control [1, 4], they pursued an optimal control framework that, briefly, resulted in a generalized set of Bellman-like equations, with the optimization being done over both the continuous and discrete actions available at each point in the hybrid (continuous cross discrete) state space. In analogy to the pure continuous and discrete cases, hybrid versions of various algorithms were developed for solving optimal hybrid control problems [7]. Among these were a boundary-value method, generalized value and policy iterations, and one based on linear programming. Some small examples were solved using these methods. The problem with these methods is the same problem that plagues all methods based on dynamic programming: the curse of dimensionality. That is, computational complexity grows quickly with the state dimension.

This paper begins to explore methods that attempt to break this curse for a class of problems. One such class is that of time optimal planning problems. In [12], Sethian introduced level set methods to solve such problems in continuous domains. The basic problem consists of start and goal sets plus a speed profile in the state space that encodes how fast one may move in the space at each point. The basic idea of level set methods is to track the evolution of the time-from-start-set curve as it propagates outward with a speed at each point given by the profile. Once this curve hits the goal set, the problem is solved. The optimal time-to-goal is the level set value of the evolved curve through the attained goal point; the optimal path is found by following the negative gradient of the surface formed by these level set curves back to the start set. See Figure 1. One particular level set method, the so-called *fast marching* method, is a particularly efficient algorithm for solving such problems. Branicky and Hebbar [5] proposed an extension of this method to solve time optimal hybrid control problems that allowed a finite number of continuous domains (each accompanied by its own speed profile) plus arbitrary jumps between domains (with a time cost of jumping parameterized by a jump's source and destination). Here, we formalize this extension, prove its correctness, and offer some computational bounds and improvements. The fast marching algorithm holds for $\Re^n$, but Sethian restricts himself to two and three dimensions for notational reasons [14]. Though notationally cumbersome, in this paper we explicitly state our extension in $\Re^n$. There is related work in robotic obstacle avoidance [17], as well as some work in extending level set methods to deal with the evolution of Hamilton-Jacobi equations [16].

[1] Supported by the National Science Foundation, under grant number DMI97-20309. Correspondence: 10900 Euclid Ave., Glennan 515B, Cleveland, OH 44142-7221. mb@ieee.org

Another way to break the curse of dimensionality is to (induce and) exploit problem structure to more quickly find optimal or near-optimal solutions. One such way to induce structure is to formulate a finite set of control laws or policies or *behaviors*, that prescribe a primitive action to be taken at each point of (a subset of) the state space. Now, control is done by switching among the indexed set of behaviors. See Figure 2. This approach is related to recent work in the area of reinforcement learning [9, 15], where researchers have introduced the notion of *temporally-abstract actions* to solve Markov Decision Problems in a more efficient manner. Our idea of behavioral programming actually extends this notion (and the related results by analogy) to the case of continuous and hybrid problems. Thus, we investigate behavioral programming techniques as a means to more efficiently generate near-optimal solutions to hybrid system control problems.

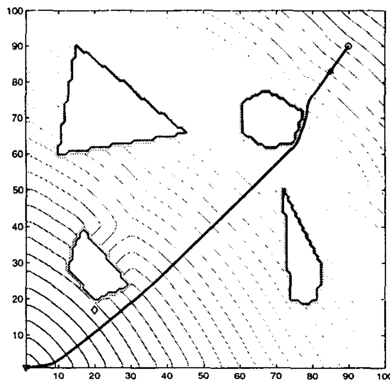

Figure 1: Level set methods solve a navigation problem

In review, the behavioral programming [3] method starts by assuming families of controllers with guaranteed stability and performance properties. These constituent controllers can then be automatically combined on-line using graph-theoretic algorithms so that higher-level dynamic tasks can be accomplished without regard to lower-level dynamics or safety constraints. The process is analogous to building sensible speech using an adaptable, but predictable, phoneme generator. Such problems can be cast as optimal hybrid control problems as in [1] as follows:

- Make N copies of the continuous state-space, one corresponding to each behavior, including a copy of the goal region in each constituent space.
- Use as dynamics in the ith copy of the state space the equations of motion that result when the ith behavior is in force.
- Allow the controlled jump set to be the whole state space; disable autonomous jumps. Impose a small switching cost.
- Solve an associated optimal hybrid control problem, e.g., penalize distance from the goal.

The result (if the goal is reachable) is a switching between behaviors that achieves the goal. See Figure 2.

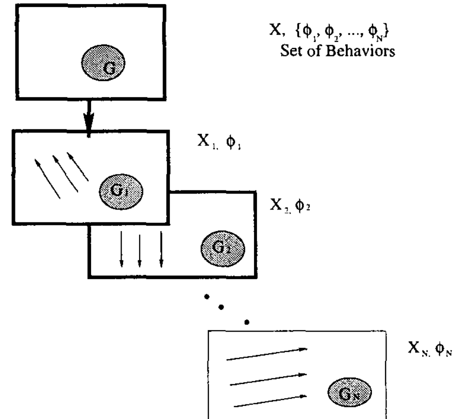

Figure 2: Behavioral Programming

The purpose of this work is to present computational methods for solving hybrid system control problems. We first review the fast marching level set method, then extend it to the hybrid setting case. We solve a stair climbing example and recall a constructive proof of the algorithm's correctness. Because the fast marching method is an approximate numerical method, we also discuss ways to obtain upper and lower bounds of the true solution. Our work also covers the behavioral programming area. We do some experiments to solve a specific problem in robot assembly: the peg-in-hole problem. We show the abstraction of the behaviors and try several strategies to solve it optimally.

The paper is organized as follows. Section 2 introduces the hybrid systems extension of the fast marching level set method, the proof, the bounds, the improvement. Section 3 demonstrates solving hybrid systems problems using the idea of behavioral programming. Section 4 presents conclusions.

2 Fast Marching for Hybrid Systems

This section describes an approach to solving optimal hybrid control problems using level set methods. Specifically, We have extended Sethian's fast marching algorithm to solve for time optimal paths for hybrid systems [5].

2.1 Problem Setup

The formal setup of the hybrid problem is as follows:

- State space: $S = \bigcup_{q=1}^{N} \{q\} \times X_q$. $X_q \subseteq \Re^{n_q}$. The state is described as the pair of discrete mode and the continuous state. There are N modes, and in the numerical algorithm, we will assume that the

continuous state spaces are partitioned into grids. In each constituent state space, X_q, we have a grid of points with spacing $\triangle x_{1,q}, \ldots, \triangle x_{n_q,q}$ and any point contained in a cell given by a discrete index given by $\mathbf{i}_q=(i_1, i_2, \ldots, i_{n_q})$. In a very special case, $X_q \equiv X$ for all $q \in 1, 2, \ldots, N$.

- Speed function: $F : S \rightarrow \Re^+$. For each mode, for each grid point, there is a corresponding speed value $F_{q,\mathbf{i}_q}$.
- Starting set: Start contour set $\Gamma \subseteq S$. The time of each start set point is set to be zero.
- Goal set $G \subseteq S$.
- Time function: $T : S \rightarrow \Re^+$. The time at which the boundary crosses this point.
- Jump function: $J : S \rightarrow 2^S$. For each $s \in S$, $J(s)$ is a subset of S which are the *jump successors* or destination points of s.
  Hence, the neighbors of a point $p \in S$ are defined as $\mathbf{N}(p)=$ $\{2n_q$-connectivity neighbors, in X_q if $p \in \{q\} \times X_q\} \bigcup J(p)$. In addition, for $s \in S$, we define its *jump predecessors* to be the set $\mathbf{P}(s) = \{p \mid s \in J(p)\}$.
- Jump delay function: $C : \bigcup_{s \in S} S \times J(S) \rightarrow \Re^+$. If a jump occurs from a state in one mode to a state in another mode, a jump delay (or jump cost) is incurred.

2.2 The Algorithm

We proposed a fast marching algorithm for solving optimal hybrid system control problems as follows [5].

1. Initialize:
   (a) Alive points: All start points are tagged as alive points and the value of T for those points is set to 0.
   (b) Narrow band points: $NB = \{p \in S \mid \exists s \in S, s$ is alive, such that $p \in N(s)$ and p is not alive$\}$. The time of arrival, T, for each of these points is computed according to the method described in 2(d) below, after setting the values of faraway points as in (c).
   (c) Faraway points: All other grid points are tagged as faraway, with T equal to ∞.
2. Marching forward:
   (a) Begin loop: Find $q, \mathbf{i}_q$ of the narrow band point with the smallest value of T.
   (b) Tag the point as an alive point and remove it from the narrow band list.
   (c) Tag its neighbors as narrow band points if they are either faraway or narrow band points. Remove neighbors in faraway list, putting them in the narrow band list.
   (d) Compute the values of T for the neighbors: For each neighbor point $p = (q, x_q)$, first we can get an evaluation value T_1 from its $2n_q$-connectivity neighbors in the same mode by solving for the largest possible solution to the quadratic equation:

$$\sum_{j=1}^{n_q} \max(D_{\mathbf{i}_q}^{-x_{j,q}} T, -D_{\mathbf{i}_q}^{+x_{j,q}} T, 0)^2 = 1/F_{\mathbf{i}_q}^2,$$

   where

$$D_{\mathbf{i}_q}^{\pm x_{j,q}} = \pm \frac{T_{(i_1,\ldots,i_{j-1},i_j \pm 1, i_{j+1},\ldots,i_{n_q})} - T_{\mathbf{i}_q})}{\Delta x_{j,q}}$$

   Then we obtain another evaluation T_2 from its jump predecessors, i.e., the sum of the arrival time of its corresponding points in another mode and the jump delay to the current mode: $T_2 = \min\{T_s + C(s,p) \mid s$ is alive and $p \in J(s)\}$. The recomputed value for point p is $T_p = \min(T_1, T_2)$.
   (e) Return to step (a).
3. Stopping criterion: One goal point is alive.

Once the propagation is complete, the time optimal path is determined through a backward trajectory from the goal point to the start point that follows the negative gradient. In each discrete state, this trajectory is orthogonal to the level set contours. If a jump successor, s, is hit in the process, we examine its predecessors. If the least of their times of arrival plus jump costs is equal to that of s, we follow the jump (backwards) to that predecessor, from which the process continues. A proof of the algorithm's correctness is in [6].

2.3 Solving Hybrid Problems by Fast Marching

We now present a simple hybrid example using the extended fast marching method. Very similar examples and their solutions appeared in [5, 6], which the reader may consult for more details. In the Stair Climbing Problem, we consider a start point in one discrete state and a goal point in some other discrete state, with transitions from one discrete state to another being explicitly defined at a specific subset of locations. To explain the concept, let us consider four floors of a building as four discrete states. See Figure 3, with floors numbered 1–4 from top to bottom and left to right. Each floor is connected to its neighboring floors by stairs, described by their sources (stars in floor i) and destinations (diamonds in floor $i+1$). The problem is to find the time optimal path from the entrance (triangle at the lower left-hand corner of floor 1) to the goal state (circle in floor four).

Each floor also has obstacles in it. We assume a nominal speed of unity, except within the obstacles, where it is set to a very low number (0.0001). This defines the continuous dynamics for each discrete state. The

stairs represent the discrete transitions between these states, where a jump relation defines the time of traversal. The stairs are assumed to be free of obstacles and hence have constant delays associated with them (1.0).

The grid point corresponding to the building entrance is the one from which the wave front propagation starts; it continues on that level following the original fast marching algorithm. As soon as the wave front hits a stair, however, we have a hybrid problem. The front must enter the next level and the propagation must continue on both of these levels. Therefore, we use our extension, as described above. That is, as long as a grid point does not have a stair to any of the other floors, it is considered to have only four neighbors. Otherwise, suppose a grid point, say p, has a stair connecting to the next floor at an attachment point there, say s. In this case, s is the jump successor of p, p is the jump predecessor of s, and the cost of taking the stair from p to s is given by the (in this case, constant) jump delay function. The level sets and the optimal solution are shown in Figure 3.

In [5, 6], we also studied a Gear Switching Problem, wherein switching between discrete states can occur everywhere on the grid (to another discrete state of gear, but maintaining the same continuous state).

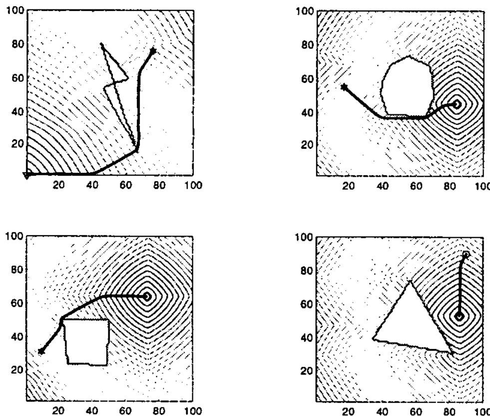

Figure 3: The Stair Climbing Problem

2.4 Upper and Lower Bounds for Marching

Our fast marching algorithm provides a numerical approximation to the continuous differential equation in each mode of the hybrid problem. Now we focus our attention on deriving approximate solutions that are upper and lower bounds of the true analog problem. The arrival time of each grid point is calculated based on the speed profile, and we assume that we know the propagation speed at each grid cell center. If we use for that value an upper bound of the analog speed function in that whole grid cell, instead of the speed function sampled at that grid point, to perform the fast marching calculation, we can obtain a lower bound of the arrival time at each grid point (because the arrival time is inversely proportional to the speed at each point). Similarly, if we use a lower bound of the speed function over the cell instead of the sample at each grid point to do the fast marching calculation, we obtain an upper bound of the arrival time approximation to each grid point.

Based on the above idea, we can compute upper and lower bounds for the Gear Switching Problem of [6]. The optimal arrival time from the start point to the goal point and the upper and lower bounds for different grid sizes are compared in Figure 4. We can see that the upper and lower bounds converge to the solution of the real problem (defined piecewise continuously on a 100×100 grid) as the number of grid cells grows.

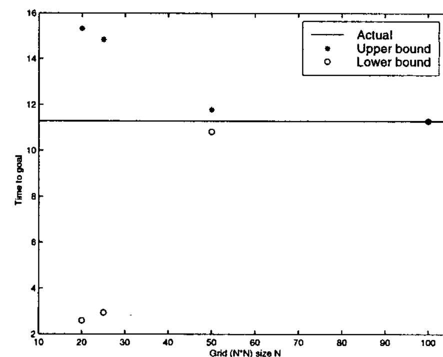

Figure 4: Bound comparison among different grid sizes

3 Behavioral Programming

In a hybrid system, the finite number of constituent continuous dynamics can be viewed as *behaviors* of the system, and the discrete dynamics can be viewed as the switching and selecting among all these behaviors [3]. See Figure 2. Recently, there has also been a lot of research on capitalizing on the idea of "behaviors" in the reinforcement learning literature using a *Markov Decision Process (MDP)* framework. See [11], on which the rest of this Section is based, for reference.

First, let us review a useful concept: the state sequence, Ω [10, 11]. Let $h_{t,T} = s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \ldots, r_T, s_T$ be the history sequence from time $t \leq T$ to time T, and let Ω denote the set of all possible histories in the given MDP (with state s_t in state space S, action a_t in action set A, and reward r_t given by a reward function $R : S \times A \rightarrow \Re$).

A temporally abstract action, or *behavior*, is a triple, $b = \langle I, \pi, \beta \rangle$, as follows [10, 11]:

- an input set $I \subseteq S$,
- a policy $\pi : \Omega \times A \rightarrow [0, 1]$, which describes the action that should be taken in a specified state

under this behavior,

- a stop condition $\beta : \Omega \rightarrow [0,1]$ of the behavior.

The transition model of the behavior can be represented by the *reward under the behavior* and the *state-prediction under the behavior*:

$$r_s^b = E\{r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{k-1} r_{t+k} \mid b\}, \quad (1)$$

$$p_{ss'}^b \doteq \sum_{j=1}^{\infty} \gamma^j \Pr\{s_{t+k} = s', k = j \mid b\}. \quad (2)$$

Here k is the *termination time* of the behavior.

Now one can also plan with behaviors, not just with primitive actions. To do so, we define a *policy over behaviors* [10]. First, the set of the behaviors available in state s can be represented as B_s. Now we have a behavior set $B = \cup_{s \in S} B_s$. The policy over behaviors is defined similarly as in the primitive action case. A policy μ over behavior set B is a function $\mu : S \times B \rightarrow [0,1]$. Thus, at arbitrary state $s \in S$, we select behavior b according to the probability $\mu(s,b)$. In this way, we can define the *state-value function over policy* μ:

$$V^\mu(s) = E\{r_{t+1} + \gamma r_{t+2} + \ldots + \mid \mu, s, t\}. \quad (3)$$

For any general Markov policy μ over behavior set B, its state-value function satisfies the following Bellman equation:

$$V^\mu(s) = \sum_{b \in B} \mu(s,b)[r_s^b + \sum_{s'} V^\mu(s')]. \quad (4)$$

Thus the *synchronous value iteration* (SVI) algorithm with only behaviors can be obtained as follows [11]:

- Start with arbitrary initial value function $V_0(s)$.
- Iterate the update:

$$V_{k+1}(s) \leftarrow \max_{b \in B_s}(r_s^b + \sum_{s' \in S} p_{ss'}^b V_k(s')), \forall s \in S. \quad (5)$$

- Repeat till the two consecutive values are "close."

Here we should notice that if we select one behavior b at state s, we will take the action according to that policy under this behavior b. Note that we can also obtain the corresponding policy once we calculate the optimal state-value function. The optimal state-value function with behaviors, V_B^*, is normally less than the optimal state-value function with only primitive actions V^*. However, we can often obtain V_B^* much faster than V^*.

3.1 A Simple Peg-in-Hole Problem

In robot assembly problems, some possible solutions to parts of the task may be known. We have examined actual strategies for peg-in-maze problems [8], capitalizing on the experience for these tasks. For example, there may exist controllers or abstract actions for moving the gripper to a specific home position or moving it until a certain force is encountered. For illustration herein, we now postulate four different behaviors for a two-dimensional peg-in-hole problem. We very roughly discretize the problem into a 3 by 7 grid of peg locations, with mating achieved when the peg moves to (3,4). Cf. Table 1.

	1.	2.	3.	4.	5.	6.	7.
1.	0	−1	−2	−3	−4	−5	−6
2.	0	−5	−10	0	−5	−10	−15
3.	*	*	*	0	*	*	*

Table 1: Reward under behavior b_1: "Move left"

Behavior 1, b_1. "Move left until you hit the wall or find (drop into) the hole", $b_1 = \langle I_1, \pi_1, \beta_1 \rangle$:

- $I_1 = S$, so b_1 can be invoked in any state.
- $\pi_1(\Omega_1, \text{LEFT}) = 1.0$, if Ω_1 is a state sequence which can be realized by always taking action LEFT from the initial state of Ω_1. $\pi_1(\Omega_1, \text{LEFT}) = 0.0$, otherwise.
- $\beta_1(\Omega_1) = 1.0$, if the last state of Ω_1 is in {1, 8, 11}; $\beta_1(\Omega_1) = 0.0$, otherwise.

This behavior can be represented as follows: $p_{s,1}^{b_1} = 1.0$ if $s \in \{1, 2, \ldots, 7\}$, $p_{s,8}^{b_1} = 1.0$ if $s \in \{8, 9, 10\}$, $p_{s,11}^{b_1} = 1.0$ if $s \in \{11, 12, 13, 14\}$. The reward under the behavior b_1 is shown in Table 3.3.

The other behaviors below and their rewards can be defined and computed in a similar manner: **behavior 2,** b_2, "Move right until you hit the wall or find the hole"; **behavior 3,** b_3, "Move up until you reach the first layer"; **behavior 4,** b_4, "Move down until you reach the surface of the part or the third layer".

3.2 Behavioral Prog. vs. Primitive Actions

We assume that the start state is uniformly chosen to be one of the states in the first layer. The utility value function (or state-value function) is defined as the expected sum of discounted rewards from the start state to the hole. Based on our experience of the hand-crafted results [8], we obtained several behaviors for the peg-in-hole problem. Now we give a comparison of the results of optimization using behavioral programming versus primitive actions. In this analysis, we assume the goal state is fixed. We perform planning according to the standard value iteration method. In the first case, we perform planning with only primitive actions; in the second case, we perform planning with our higher-level behaviors from Section 3.1.

Without any doubt we can get the optimal result if we perform planning only with primitive actions. However, the planning procedure becomes quite long as the number of states becomes large. In cases where the optimal process is impractical, we can apply behavioral programming. In particular, we can take advantage of the results of Section 3 to perform planning with be-

haviors that accelerates the optimization process. We have defined four available behaviors in Section 3.1. We should also notice that primitive actions themselves can also be considered as very simple behaviors. Thus, we can plan with 8 behaviors (4 behaviors defined in Section 3.1 plus the 4 primitive actions). The relative *Root Mean Square Error (RMSE)* compared to the optimal utility value is shown in Figure 5. In this case, the optimal utility value function was calculated by planning with only primitive actions. From Figure 5, we notice that planning with behaviors can accelerate planning, but may not converge to the optimal solution. Thus, behavioral programming poses a tradeoff between suboptimality and computation time.

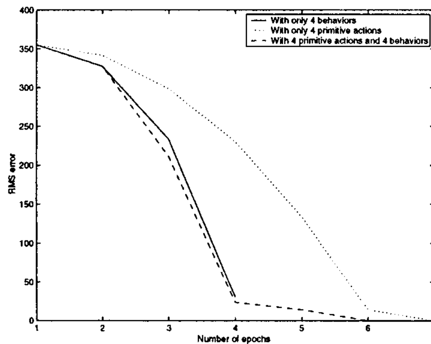

Figure 5: Behavioral programming vs. primitive actions

4 Conclusions

Hybrid systems includes both continuous dynamics and discrete events. This paper was aimed at using level set methods and behavior programming to solve hybrid control problems. Generally it is not easy to solve such hybrid system control problems analytically. Herein, we study the application of computational methods for solving them. In particular, we extend a computational method that has been found useful to solve continuous time optimal planning problems to the hybrid setting, opening up a class of hybrid optimal control problems to more efficient solution. We also use an idea that we call "behavioral programming" to induce a hierarchical structure into hybrid control problems that can be exploited to more quickly find (near-optimal) solutions.

Specifically, we first reviewed our extension of the fast marching level set method to the hybrid setting, then provided upper- and lower-bound approximations of the analog solution. We also presented some results for a new solution technique that we call "Behavioral Programming". Two example hybrid problems were discussed. A stair-climbing problem was solved using fast marching level set method; a "peg-in-maze problem" was solved using behavior programming. Our results can be used solve more complicated hybrid systems.

References

[1] M.S. Branicky, *Studies in Hybrid Systems: Modeling, Analysis, and Control*, Sc.D. Dissertation, Electrical Engineering and Computer Science Dept., Massachusetts Institute of Technology, June 1995.

[2] M.S. Branicky, Multiple Lyapunov functions and other analysis tools for switched and hybrid systems, *IEEE Trans. Automatic Control*, **43**(4):475–482, 1998.

[3] M.S. Branicky, Behavioral programming: Enabling a "middle-out" approach to learning and intelligent systems, *Proc. IFAC Intl. Symp. on AI in Real-Time Control*, Grand Canyon Natl. Park, AZ, Oct. 1998.

[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 and R. Hebbar, Fast marching for hybrid control, *Proc. IEEE Computer-Aided Control System Design*, Kona, HI, Aug. 1999.

[6] M.S. Branicky, R. Hebbar, and G. Zhang, A fast marching algorithm for hybrid systems, *Proc. IEEE Conf. on Decision and Control*, Phoenix, AZ, Dec. 1999.

[7] 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.

[8] S.R. Chhatpar and M.S. Branicky, A hybrid systems approach to force-guided robotic assemblies, *Proc. IEEE International Symp. on Assembly and Task Planning*, Porto, Portugal, July 1999.

[9] L.P. Kaelbling, M.L. Littman, and A.W. Moore, Reinforcement learning: A survey, *Journal of Artificial Intelligence Research*, Vol. 4, 1996.

[10] D. Precup and R.S. Sutton, Multi-time models for reinforcement learning, *Proc. ICML'97 Workshop on Modeling in Reinforcement Learning*, 1997.

[11] D. Precup, R.S. Sutton, and S. Singh, Theoretical results on reinforcement learning with temporally abstract behaviors, *Proc. Machine Learning: ECML-98. 10th European Conf. on Machine Learning*, Chemnitz, Germany, pp. 382–393, Springer, 1998.

[12] J.A. Sethian, *Level Set Methods*, Cambridge U. Press, 1996.

[13] J.A. Sethian, A fast marching level set method for monotonically advancing fronts, *Proc. Nat. Acad. Sci.*, **93**(4):1591–1595, 1996.

[14] J.A. Sethian, Fast marching methods, *SIAM Review*, **41**(2): 199–235, 1999.

[15] R.S. Sutton and A.G. Barto *Reinforcement Learning*, MIT Press, Cambridge, MA, 1998.

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

[17] K.I. Trovato, *A* Planning in Discrete Configuration Spaces of Autonomous System*, Ph.D. Dissertation, University of Amsterdam, September 1996.