

On-line Techniques for Behavioral Programming

Michael S. Branicky¹, Tor Arne Johansen², Idar Petersen,^{*†} Emilio Frazzoli³
mb@ieee.org, tor.a.johansen@itk.ntnu.no, ip@itk.ntnu.no, frazzoli@mit.edu

Abstract

Many control problems of interest can be cast as an optimal hybrid system control problems, wherein an objective function represents some global goals and the input at each time instant is a choice among a finite set of control laws. There are many approaches to solving such problems in the literature, all based on dynamic programming in some form or another, and all suffering from overwhelming computational complexity. Herein, we attempt to lower this complexity by examining techniques that take advantage of the underlying properties of the individual controllers among which we are switching. We call this process “behavioral programming,” since we are now attempting to perform dynamic programming at the more abstract level of behaviors of the constituent systems. We present our paradigm and discuss two arenas of its use: motion planning for autonomous agents and LQR with state and input constraints. Applications to helicopter and wheel slip control are used to illustrate problem-solving in each of these arenas, respectively.

1 Behavioral Programming

A *hybrid system* consists of a finite number of constituent dynamical systems, or *behaviors*, plus rules for switching and selecting among them [2]. Thus, there are discrete or higher-level switching dynamics (which can be captured as an automaton or graph structure) combined with the lower-level dynamics (usually continuous-state and given by differential or difference equations). An optimal control theory for such systems has been developed (see, e.g., [3, 5, 4]).

Unfortunately, such solutions are based on dynamic programming and inevitably suffer from a curse of dimensionality. Lately, we have been intrigued with the possibility of speeding up the solution to such problems by considering techniques that take advantage of the underlying properties of the individual behaviors among which we are switching. We call this pro-

cess “behavioral programming” [6], since we are now attempting to perform dynamic programming at the more abstract level of behaviors of the constituent systems.

Related to this viewpoint, is the idea of “temporally abstract behaviors” developed in the reinforcement learning literature for Markov decision processes (MDPs). See [18] for an overview.

It is possible to explain behavioral programming in a continuous time framework (see [6]). Indeed, the problem arenas we explore in Sections 3 and 4 possess underlying continuous-time models. For simplicity, however, we present the overall idea of behavioral programming in the discrete, MDP framework, following [19].

In the MDP framework, an agent interacts with the environment at a discrete, lower-level time scale. The system description includes: a discrete state space, S ; an action space, A ; and a reward function $r : S \times A \rightarrow \mathfrak{R}$

The one-step transition model can be represented by a one-step transition probability and a one-step expected reward.

$$p_{ss'}^a = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \quad (1)$$

$$r_s^a = E\{r_{t+1} \mid s_t = s, a_t = a\} \quad (2)$$

for all $s, s' \in S$ and $a \in A$.

A *Markov Policy* is defined as a mapping from states to probabilities when taking each specific action in the action space, $\pi : S \times A \rightarrow [0, 1]$. The agent’s objective is to learn an *optimal Markov policy*, which is one maximizing the expected sum of discounted future rewards from each state s if the system takes action according to this policy. First, we examine the value of following a particular policy π :

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

Here, $\pi(s, a)$ is the probability with which the policy π chooses action $a \in A$ in state s and $\gamma \in [0, 1)$ is a *discount factor*. We have thus obtained the *state-value function* (or *utility value function*) under the policy π : V^π .

The *optimal state-value function* gives the value of a

¹Electrical Engineering and Computer Science Dept., Case Western Reserve U., Cleveland, OH 44106-7221 U.S.A.

²Engineering Cybernetics Dept., Norwegian University of Science and Technology, N-7491 Trondheim, NORWAY

³Aeronautics and Astronautics Dept., Massachusetts Institute of Technology, Cambridge, MA 02139 U.S.A.

state under an optimal policy:

$$V^*(s) = \max_{\pi} V^{\pi}(s) = \max_{a \in A} \left[r_s^a + \gamma \sum_{s'} p_{ss'}^a V^*(s') \right] \quad (4)$$

The temporally abstract action can be represented as a *Behavior*, which is a triple, $b = (I, \pi, \beta)$, as follows: an input set, $I \subseteq S$; a policy, π , which describes the action that should be taken in a specified state under this behavior; and a stop condition, β , of the behavior.

The transition model of the behavior can be represented by the reward and state prediction functions, both under the behavior b and from state s :

$$\begin{aligned} r_s^b &= E\{r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{k-1} r_{t+k} \mid b\}, \\ p_{ss'}^b &= \sum_{j=1}^{\infty} \gamma^j Pr\{s_{t+k} = s', k = j \mid b\} \end{aligned}$$

Here k is the termination time and $s' \in S$.

Now, we can plan with only behaviors, and not with primitive actions. We can define the *policy over behaviors*. 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 the primitive action case. A policy over behavior $\mu : S \times B \rightarrow [0, 1]$. So at any arbitrary state $s \in S$, we can select (proper) behavior b according to the probability $\mu(s, b)$. In this way, we can define *state-value function over policy* μ :

$$V^{\mu}(s) = E\{r_{t+1} + \gamma r_{t+2} + \dots \mid \mu, s, t\} \quad (5)$$

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

$$V^{\mu}(s) = \sum_{b \in B} \mu(s, b) \left[r_s^b + \sum_{s'} V^{\mu}(s') \right] \quad (6)$$

Then the *synchronous value iteration* (SVI), with only behaviors, can be obtained as follows:

- start with arbitrary initial values $V_0(s)$
- iterate the update:

$$V_{k+1}(s) \leftarrow \max_{b \in B_s} \left(r_s^b + \sum_{s' \in S} p_{ss'}^b V_k(s') \right), \quad \forall s \in S \quad (7)$$

- Repeat until two consecutive values are close enough

Here, we should notice that if we select one behavior b at state s , we will take the action according to the policy under this behavior b .

The optimal state-value function with behaviors V_B^* is normally less than the optimal state-value function with only primitive actions V^* . But, we can obtain V_B^* with a faster procedure than we can obtain V^* .

The key of course is to pick the **right** behaviors: ones which lead to performance that is near the optimal (for primitive actions) and whose rewards and policies are readily computable. In our application arenas below, we choose them to be trim trajectories in motion planning and linear controllers for constrained LQR.

2 Motion Planning for Autonomous Vehicles

One of the basic problems which have to be faced by autonomous vehicles or moving robots is the generation and the execution of a motion plan, that enables the robot to move to some desired place to perform a given task, while avoiding collisions with obstacles, or other undesired behaviors. Moreover, we would like the generated motion plan to be compatible with the system dynamics: this is particularly true when the exploitation of the vehicle's maneuvering capabilities is key to the successful completion of the mission.

In many cases of interest, including for example autonomous aerospace vehicles, the state space is typically large: in the simple case in which the vehicle is modeled as a rigid body subject to commanded forces and torques, the state space has dimension 12. If additional dynamics are considered (e.g. engine, actuator, or structural dynamics), the dimension of the state space can be even larger. In such cases the "curse of dimensionality" makes the solution of motion planning problems in such large-dimension spaces computationally intractable.

An alternative approach is represented by the formulation of the system dynamics, and hence of the motion planning problem, in what can be regarded as the *maneuver space* of the vehicle[10, 11]. Such an approach entails the definition of a hybrid control architecture (a *hybrid automaton*), based on quantization of the system dynamics into a library of feasible "trajectory primitives", or *behaviors*. See Figure 1. There, each node represents a trim trajectory (cf. behavior) of the helicopter, and transitions between edges are maneuvers driving the craft from the initial trim trajectory to the next chosen trim trajectory. Trajectories are composed by traversing the states of the machine, i.e., switching among the behaviors of the system. Control actions include the dwell times in each trim trajectory and the times and the destinations of the transition maneuvers.

2.1 Motion Planning Algorithm

We begin by considering the (obstacle-free) problem of finding the control input that, given a target *equilib-*

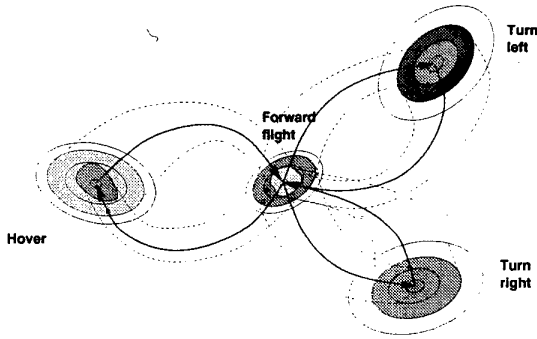


Figure 1: Simplified Robust Hybrid Automaton

rium point x_{eq} , minimizes the total cost:

$$J(x_0, x_{eq}) = \int_{t_0}^{t_f} g(x, x_{eq}, u) dt + \Gamma(x(t_f), x_{eq}, t_f)$$

for some initial conditions x_0 , under the dynamics constraints (and possibly state and control constraints).

The solution to an optimal control problem in the free space thus provides us with a control policy π that ensures that the system is driven towards an equilibrium point, effectively parameterized by all configurations of the agent. Additionally, we notice that, once a cost function has been computed for all states, it can be used as a meaningful measure of the “distance” from points in the state space to equilibrium points at arbitrary locations (under the invariance hypothesis; see [11]).

Based on the optimal control policy (over behaviors) so derived, and on the fundamental ideas behind probabilistic roadmaps in robotics [15, 17], we have defined a motion planning algorithm that determines a time-parameterized sequence of randomized “attraction points” $x_{eq}(t)$ that effectively steers the system to the desired configuration while avoiding obstacles.

In the following, we will present a brief outline of the algorithm. More details on the algorithm, and an analysis of its completeness and performance can be found in [9]. Starting with a node representing the initial condition, we build a tree by iteratively adding new “milestones”, which are connected to the tree by a feasible trajectory segment. Each new milestone is generated through the generation of a random equilibrium state x_{rand} ; the control policy $\pi(\cdot, x_{rand})$ is applied to a randomly chosen node of the current tree, and if the ensuing propagated trajectory is feasible, x_{rand} is added to the tree. The milestones generated according to the above procedure can be regarded as *primary* milestones. In the following, we will assume that x_{rand} is generated from a uniform distribution on the subset of \mathcal{C} defining our workspace.

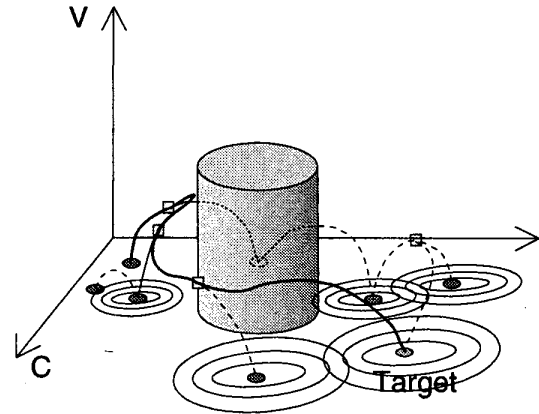


Figure 2: Example of generated roadmap (projected on \mathcal{X}). Primary milestones are marked as circles, and secondary milestones as squares

As it can be easily recognized, the algorithm outlined above consists of jumps from equilibrium point to equilibrium point, and as such cannot be expected to provide impressive performance, especially in terms of time. In order to increase the performance, we will introduce the following step: at random points along the trajectory to the new randomly generated point, we add $n_s \geq 1$ additional, *secondary*, milestones. Secondary milestones are likely to be at points in the state space that are “far” from equilibria. From secondary milestones we can apply the control policy to the destination $\pi(\cdot, x_f)$: if the resulting trajectory is feasible, we have solved the feasibility problem. In this case, we climb the tree back towards root, updating the estimates on the upper bound on the cost-to-go. Both in the case in which a feasible trajectory is found, and in the case in which a collision is detected, the secondary milestones are added to the tree, and can be selected as the starting point for later iterations. Note that each secondary milestone, by construction, will have a primary milestone in a child subtree (see Fig. 2).

2.2 Application: Autonomous Helicopter

In this section, we will present simulation results for a test case involving a small autonomous helicopter. The simulations are carried out on a fully non-linear helicopter simulation, based on a widely used minimum-complexity model [16]. The motion planning algorithms operating on the hybrid automaton structure [10, 11] are complemented by the control law presented in [12] to ensure tracking of the reference trajectory. The cost function used in all the examples is the total time needed to go to the destination (we are solving a minimum-time problem). The algorithm was implemented in C on a 300 MHz Pentium II computer.

A first example involves navigating the helicopter through a set of obstacles represented as spheres in

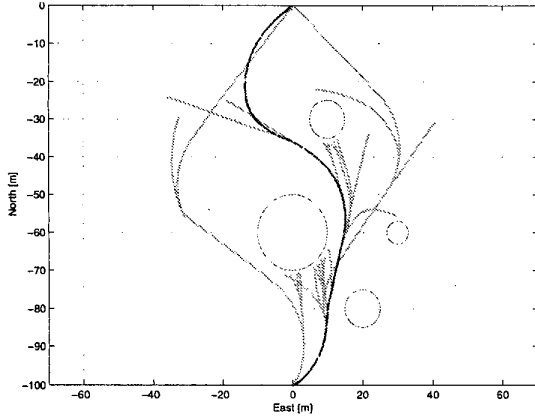


Figure 3: Example 1: trace of the trajectory tree, and best trajectory found

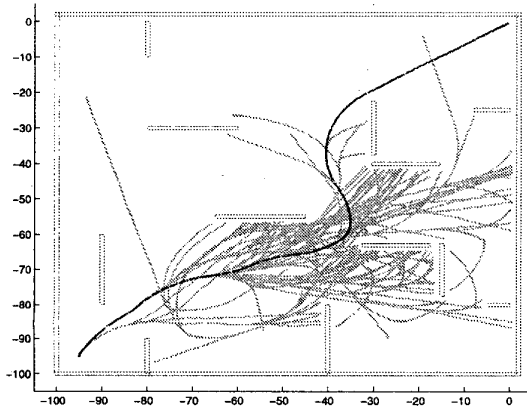


Figure 4: Example 2: trace of the trajectory tree, and best trajectory found

the configuration space. This case was tackled very easily by our algorithm, both in the case of fixed obstacles and in the case of moving obstacle. A feasible, almost optimal trajectory was found in real-time, while the helicopter was already in motion. See Figure 3. Navigation through a maze (Figure 4) was slightly more problematic, since a feasible solution is found several seconds into the flight. However, we must remark that the algorithm we presented always retain safety guarantees, by ensuring that all the leaves of the tree (i.e. the terminal states) are safe, primary milestone. Thus, even in situations where the helicopter is flying “aggressively”, we are retaining the capability to “stop and think” in a safe location, if need arises.

3 LQR with State and Input Constraints

The problem of Linear Quadratic Regulation with Constraints (LQRC) is studied in [14]. Therein, they pro-

pose a suboptimal solution strategy, where an approximation of the optimal cost function is utilized and with restrictions on the switching that is allowed between active constraint sets for a given prediction horizon. Specifically, the technique involves a search over the set of behaviors given by linear feedback controllers. The technique tends to require less amounts of real-time computer memory and processing power than other proposed solutions (based on dynamic programming), since all possible linear feedback controllers (behaviors) have been calculated off-line.

3.1 Explicit Feedback Solution

Consider the continuous-time LTI system $\dot{x} = A_c x + B_c^c u_c + B_c^d u_d$ and its sampled discrete-time version

$$x(t+1) = Ax(t) + B^c u_c(t) + B^d u_d(t) \quad (8)$$

where $x \in R^n$, $u_d \in U_d \subset R^r$ and $u_c \in R^{r_c}$. The elements of u_d are discrete control inputs that are restricted to take values in a finite set U_d , while the elements of u_c are continuous control inputs that take real values, possibly restricted to a convex subset $U_c \subset R^{r_c}$. For convenience, denote $u(t) = (u_c^T, u_d^T)^T$ and $B_c = (B_c^c, B_c^d)$, $B = (B^c, B^d)$, such that $\dot{x} = A_c x + B_c u$ and its sampled discrete-time version $x(t+1) = Ax(t) + Bu(t)$.

The optimal constrained LQ feedback controller minimizes the infinite horizon quadratic cost

$$J(u(t), u(t+1), \dots; x(t)) = \sum_{\tau=t}^{\infty} l_{QR}(x(\tau), u(\tau))$$

$$l_{QR}(x, u) = x^T Q x + u^T R u$$

subject to the constraints

$$Gx(\tau+1) \leq g, \quad u_d(\tau) \in U_d, \quad Hu_c(\tau) \leq h \quad (9)$$

for all $\tau \geq t$, where $R > 0$, $Q \geq 0$, $G \in R^{q \times n}$, and $H \in R^{p \times r}$. It is assumed that $g, h > 0$ to ensure that the origin is an interior point in the admissible region. The *optimal cost function* is defined as

$$V(x(t)) = \min_{u(t), u(t+1), \dots} J(u(t), u(t+1), \dots; x(t)) \quad (10)$$

where the minimization is subject to the dynamics of the system (8), and the constraints (9) are imposed at every time instant $\tau \in \{t, t+1, t+2, \dots\}$ on the trajectory. The cost of moving from the state $x(t)$ to the origin in an optimal manner is given by $V(x(t))$. The Hamilton-Jacobi-Bellman (HJB) equation characterizes the optimal cost function and optimal control action for the problem:

$$V(x(t)) = \min_{\substack{u_d(\tau) \in U_d, Gx(\tau+1) \leq g, Hu_c(\tau) \leq h \\ \tau \in \{t, t+1, t+2, \dots, t+N-1\}}} \left(V(x(t+N)) + \sum_{\tau=t}^{t+N-1} l_{QR}(x(\tau), u(\tau)) \right)$$

where $N \geq 1$ is some horizon, and $V(0) = 0$. In order to determine a solution, it is rewritten as follows:

$$0 = \min_{k \in \mathcal{C}} \left(\min_{\substack{\tilde{u}_c(t) \in \mathbb{R}^{r_c N} \\ L_k \tilde{u}_c = M_k x(t) + M_k^g g + M_k^h h}} (V(x(t+N)) - V(x(t)) + \sum_{\tau=0}^{N-1} l_{QR}(x(t+\tau), E_{\tau+1} \tilde{u}_c(t))) \right)$$

where the outer minimization is also subject to the input and state inequality constraints, although this is not written explicitly. The finite set \mathcal{C} enumerates all the feasible active constraint set sequences corresponding to the original inequality constraints in addition to combinations of the discrete inputs. Notice that for each $k \in \mathcal{C}$, one can compute an explicit optimal solution to the innermost optimization problem, and due to the linearity of the constraints and quadratic cost, these state feedbacks are linear. The associated feedback matrices are thus computed off-line and stored in computer memory for real-time use. The optimal constrained LQ control can be reconstructed by optimal switching between a finite number of known linear control behaviors. The solution to the optimal switching problem is now given by the discrete optimization, which can either be (partially or completely) solved off-line and stored in real-time computer memory in terms of a state space partition, or solved in real-time using discrete search, i.e. a form of behavioral programming. These aspects are discussed in [14, 1].

A remaining problem is how to compute $V(x)$. It is known from [20, 8] that $V(x(t+N)) = x^T(t+N)Px(t+N)$ provided N is sufficiently large and $P > 0$ is the solution of the algebraic Riccati equation of the unconstrained LQ problem. However, to address the computational complexity, N will typically be chosen to be smaller with resulting suboptimality.

3.2 Application: Wheel Slip Control

A wheel slip controller is the core of automotive ABS brakes. The continuous-time slip dynamics of a quarter of a car are [7]

$$\dot{\lambda} = -\frac{1}{v} \left(\frac{1}{m}(1-\lambda) + \frac{r^2}{J} \right) F_z \mu(\lambda, \mu_H, \alpha) + \frac{1}{v} \frac{r}{J} T_b$$

where λ is the slip (i.e. the normalized relative speed of the wheel and vehicle), v is the speed of the vehicle, J is the wheel moment of inertia, r is the wheel radius, $4m$ is the mass of the vehicle (assuming braking on 4 wheels), F_z is the vertical force on the tire, α is the lateral wheel slip angle, μ_H is the road/tire friction coefficient, T_b is the clamping torque (control input), and $\mu(\cdot)$ is the nonlinear tire friction curve. For control purposes, these dynamics are discretized and linearized. Furthermore, the controller contains a model of the actuator and integral action.

The inherent constraints of the problem are due to actuator saturation and rate limits. With the present control formulation the activation of some of these constraints leads to the controller switching behavior. However, the controller may also switch behavior as a consequence of external variables and events, such as (smooth) changes in the vehicle speed v (this is essentially gain scheduling), driver commands, lateral wheel slip angle α and instantaneous changes in the road/tire friction coefficient μ_H due to driving on a heterogeneous surface. In Figure 5, we show simulation results using a simulator that includes full body dynamics, actuator dynamics, sensor noise, observers and Kalman filters for non-measured signals and computer communication delays. The objective is to control the slip λ to $\lambda^* = 0.14$, which gives close to optimal braking efficiency without too much loss of steer-ability. We observe several control behavior switches during the simulation scenario. The actuator rate constraints are active initially, until a LQ controller determines the control input. At time $t = 1.5$ s, the road/tire friction coefficient changes from $\mu_H = 0.8$ to $\mu_H = 0.4$ and it can be seen that the wheels then lock for a short period (since λ is close to 1). In order to unlock the wheels the controller switches behavior (actuator rate constraints are activated) in order to decrease the clamping force, and to reduce the slip. The small transients after $t \approx 4.5$ s and $t \approx 5.3$ s are caused by controller switching due to gain scheduling on the vehicle speed. We observe that when the speed becomes very low, the problem is singular and the controller is switched off.

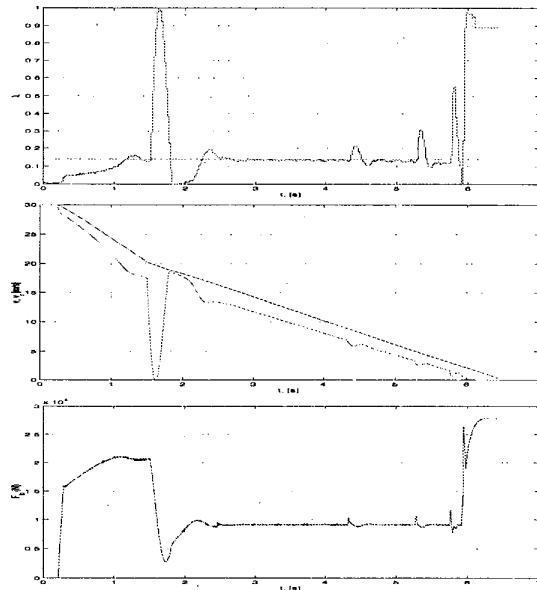


Figure 5: Simulation of braking.

4 Conclusions and Future Work

We are currently exploring how heuristic search techniques like A^* [13, 21, 22] and branch-and-bound can be used to speed up the on-line techniques described above even further. Basically, our idea is to use A^* search at the abstract level of behaviors instead of at the primitive action level. Below, we quickly review the work at the primitive action level by [13, 21].

The framework presented in [13] starts with optimal control problems defined in continuous state spaces and with continuous controls, state and input constraints, and obstacle and goal regions (in state space). Through discretization in time and space, he converts these to finite-state decision problems (though of immense size). He then performs the search for minimum-cost trajectories (from a given start point to a goal region) using a flavor of the heuristic search technique A^* . Specifically, he uses the A_ϵ^* algorithm, which allows a controlled loss of optimality—to within a factor of $(1 + \epsilon)$ —with the benefit of accelerating the search. In [21], these ideas are applied to analyze the effects of using on-line optimization to obtain a sub-optimal controller guaranteed to asymptotically stabilize systems of linear difference equations. In our applications, the specialized sub-structure of each constituent behavior (trim trajectories and linear systems, respectively) will be likewise exploited to quickly produce bounds on incremental cost (of each behavior) as well as overall cost-to-go.

References

- [1] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, The explicit solution of model predictive control via multiparametric quadratic programming, in *Proc. American Control Conf., Chicago*, 2000, pp. 872–876.
- [2] M.S. Branicky, *Studies in Hybrid Systems: Modeling, Analysis, and Control*, Sc.D. Dissertation, Electrical Engineering and Computer Science Dept., M.I.T., June 1995.
- [3] 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.
- [4] M.S. Branicky, R. Hebbbar, and G. Zhang, A fast marching algorithm for hybrid systems, *Proc. IEEE Conf. on Decision and Control*, Phoenix, AZ, December 1999.
- [5] M.S. Branicky and S.K. Mitter, Algorithms for optimal hybrid control. In *Proc. IEEE Conf. Decision and Control*, New Orleans, LA, pp. 2661–2666, Dec. 1995.
- [6] 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.
- [7] M. Burckhardt, *Fahrwerktechnik: Radschlupfregel-systeme*, Vogel-Verlag, Germany, 1993.
- [8] D. Chmielewski and V. Manousiouthakis, On constrained infinite-time linear quadratic optimal control, *Systems and Control Letters*, vol. 29, pp. 121–129, 1996.
- [9] E. Feron, E. Frazzoli, M.A. Dahleh. Real-time motion planning for agile autonomous vehicles. In *AIAA Conf. on Guidance, Navigation and Control*, Denver, August 2000.
- [10] 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.
- [11] E. Frazzoli, M.A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicle motion planning. Tech. Report LIDS-P-2468, M.I.T., 1999. Submitted to *IEEE Trans. Automatic Control*.
- [12] E. Frazzoli, M.A. Dahleh, and E. Feron. Trajectory tracking control design for autonomous helicopters using a backstepping algorithm. In *American Control Conf.*, Chicago, IL, 2000.
- [13] Allon Guez. Heuristically enhanced optimal control. *Proc. IEEE Conf. on Decision and Control*, Athens, Greece, pp. 633–637, 1986.
- [14] T. A. Johansen, I. Petersen, and O. Slupphaug, Explicit suboptimal linear quadratic regulation with input and state constraints, *Proc. IEEE Conf. Decision and Control*, Sydney, 2000.
- [15] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, **12**(4):566–580, 1996.
- [16] T.J. Koo and S. Sastry. Output tracking control design of a helicopter model based on approximate linearization. In *IEEE Conf. on Decision and Control*, 1998.
- [17] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Tech. Report 98-11, Iowa State University, Ames, IA, Oct. 1998.
- [18] R.E. Parr, *Hierarchical Control and Learning for Markov Decision Processes*, Ph.D. Dissertation, Electrical Engineering and Computer Science Dept., University of California at Berkeley, 1998.
- [19] D. Precup, R.S. Sutton, and S. Singh, Theoretical results on reinforcement learning with temporally abstract behaviors, In *Machine Learning, Proc. ECML-98. 10th European Conf. on Machine Learning*, Chemnitz, Germany, April 1998, pp. 382–393. Springer Verlag.
- [20] M. Sznaier and M. J. Damborg, Suboptimal control of linear systems with state and control inequality constraints, in *Proc. 26th IEEE Conf. Decision and Control*, 1987, pp. 761–762.
- [21] M. Sznaier and M.J. Damborg. Heuristically enhanced feedback control of constrained discrete-time linear systems, *Automatica*, **26**(3):521–532, 1990.
- [22] K.I. Trovato, *A* Planning in Discrete Configuration Spaces of Autonomous System*, Ph.D. Dissertation, University of Amsterdam, September 1996.