Title
A Design Framework For Hierarchical, Hybrid Control

Authors
Lygeros, John
Godbole, Datta N.
Sastry, Shankar

Publication Date
1997

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

A Design Framework for Hierarchical, Hybrid Control

John Lygeros
Datta N. Godbole
Shankar Sastry

California PATH Research Report
UCB-ITS-PRR-97-24

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

May 1997

ISSN 1055-1425

A Design Framework for Hierarchical, Hybrid Control*

John Lygeros, Datta N. Godbole and Shankar Sastry

Intelligent Machines and Robotics Laboratory
University of California, Berkeley
Berkeley, CA 94720
lygeros, godbole, sastry@eecs.berkeley.edu

Abstract

Large scale systems are common in applications such as chemical process control, power generation and distribution and transportation systems, among others. Of particular interest are multi-agent, scarce resource problems, where a large number of agents have to make efficient use of a scarce resource. Here we try to approach the problem of large scale systems from a hierarchical, hybrid control view point. Our analysis is based on a new hybrid dynamical system formulation, that allows us to model large scale systems in a modular fashion. We primarily address the problem of controller design for multi-agent systems. A design procedure is proposed that naturally leads to hierarchical, hybrid control schemes, with continuous controllers trying to optimize each agent's resource utilization at a lower level and discrete controllers resolving inter-agent conflicts at a higher level. An algorithm is presented to design the continuous controllers, as well as abstractions of their performance in terms of the discrete level. The algorithm makes use of ideas from game theory, treating the design process as a two player, zero sum game, between the controller of an agent and the disturbance generated by the actions of other agents. The resulting abstractions can be thought of as guidelines for the design of the discrete layer. If the resulting continuous controllers are used and the discrete controller satisfies the guidelines, the closed loop hybrid system is, by design, guaranteed to exhibit the desired behavior. We demonstrate our algorithm by application to the automated vehicle following problem.

1 Introduction

The focus of control theory has traditionally been on what one might call the *central control paradigm*. In this setting, the physical process to be controlled is modeled by a set of dynamic

*Research supported by the Army Research Office under grant DAAH 04-95-1-0588 and the PATH program, Institute of Transportation Studies, University of California, Berkeley, under MOU-135 and MOU-238.

equations describing the evolution of the state of the process (which can be either continuous or discrete) over time. The process can be observed through outputs (obtained from sensors) and its evolution can be influenced by inputs (applied through actuators). The task of the designer is to produce an algorithm that collects the output data, processes it centrally and applies the appropriate inputs to guide the process in the desired direction. Controllers designed in this setting have proved very effective for relatively small systems. Formal design techniques have been developed to guarantee various aspects of the system performance: stability, robustness, optimality with respect to certain criteria, disturbance rejection, etc.

Recently *large scale systems* characterized by a large, possibly dynamically changing number of inputs, outputs, and states have attracted considerable attention. Examples include chemical plants, power generation and distribution systems and transportation systems. The renewed interest in these systems has been triggered by economic and environmental considerations. The size of these businesses is such that even small changes in performance translate to large amounts of money gained or lost and/or considerable impact to the environment. At the same time, recent technological advances, such as faster computers, cheaper and more reliable sensors and the integration of control considerations in the product design and manufacturing process, have made it possible to extend the practical applications of control to systems that were too complex to control in the past.

A class of large scale systems of particular interest are *multi-agent, scarce resource systems*. Their common characteristic is that a large number of agents, equipped with sensing, communication and control capabilities, are trying to make optimum use of a congested, common resource. Examples of such systems are highway systems, (where vehicles compete for highway space-time), air traffic management systems (where aircraft compete for air space and runway space), power generation and distribution systems (where producers and consumers of power make use of the common distribution grid), computer networks, etc. The design of controllers for such systems presents a number of difficulties. The number of agents may be large and dynamically changing (planes take off and land, vehicles enter and exit the highway, etc.). Even though the state space of an individual agent may be small, the possible coupling between the agents leads to a very large number of interacting states. This coupling is even more pronounced in the presence of faults, as degraded performance of one agent may adversely effect everyone else. Moreover, the desired performance is typically given in terms of the collective behavior of all the agents (emergent behavior). Unless special measures are taken, the optimum policy for each agent need not coincide with the "common good" and compromises may need to be made. Because of the challenging problems associated with the control of multi-agent, scarce resource systems, this area has attracted considerable attention both theoretically and in application [1, 2, 3, 4, 5].

The obvious way to apply the central control paradigm to large scale systems is to design a *centralized control scheme*. In such a scheme, the information from the entire system is collected, processed centrally and commands are distributed to all the actuators. The biggest advantage of such a control scheme is that the designer can try to globally optimize the system performance. In practice, however, a centralized control scheme may be difficult to design and implement:

- The volume of information that needs to be exchanged is large. This can be a severe drawback if communicating information is expensive and/or the bandwidth is limited.

- The design process may be too complicated. Especially in multiagent systems, conflicting design goals may make it impossible to pose the controller design as an optimization problem. Even in cases where this is possible, the resulting calculation is likely to be very complex.

- The on-line computations needed to implement the controller may require unreasonable computational power. This problem is especially pronounced in multi-agent systems where the number of agents may be dynamically changing and therefore the optimal control calculations may have to be constantly updated.

- The scheme may be unreliable. The cost of hardware ultimately limits the redundancy built into the control scheme. Because of the size of the system the consequences are likely to be catastrophic if the central controller is disabled.

These difficulties (especially the complexity of the design process) have limited the practical applications of fully centralized controllers to highly structured systems, such as the scheduling of trains.

The central control paradigm can also be applied to large scale systems in a *decentralized control scheme*. Here the system is first split into subsystems. For example, in multi agent systems natural subsystems are the agents themselves. Local controllers are designed, that make use of the limited information available to each subsystem to calculate commands for the subsystem actuators. The coupling between subsystems can be taken care of locally, by making the controller more conservative and therefore more robust to the actions of the neighboring subsystems. Decentralized control schemes are easier to design. They have been shown to work well in problems where the coupling between subsystems is weak and/or the optimality of the closed loop performance is not crucial [6]. In some cases, however, proofs of performance claims are difficult to obtain, since global behavior may be hard to infer from local interactions. More importantly, the process starts to break down when one tries to push the system performance to its limit.

If a completely decentralized solution is unacceptable and a completely centralized solution is prohibitingly complex or expensive, an in-between compromise will have to be sought. The resulting control scheme will feature some form of multi level *hierarchy*, with lower levels dealing with local and higher levels dealing with global aspects of the performance. Clearly such a solution is likely to be less efficient than a centralized scheme and harder to implement than a decentralized scheme; however, it may be the only feasible choice. Different models of the physical process are likely to be needed in such a hierarchy. The lower levels need detailed models for precise local control. Typical choices are the standard, continuous models consisting of differential or difference equations. On the other hand, the reasons that make a centralized scheme impractical dictate the use of more abstract models at the higher layers. Typical choices are discrete statistical or linguistic models. The different levels of abstraction at the various levels of the controller lead to *hybrid systems*.

For multi-agent systems, a control hierarchy will involve *semiautonomous agent operation*. In this case each agent is trying to optimize its own usage of the resource and coordinates with neighboring agents if there is a conflict of objectives. It should be noted that semiautonomous agent control is naturally suited for hybrid designs. At the lower levels each

agent chooses its own optimal strategy, which will be in the form of a continuous control law. At the higher levels discrete coordination is used to resolve conflicts.

As the central control paradigm breaks down in this case, there is a *lack of formal mathematical tools* for the design and analysis of hierarchical controllers for large scale systems. Through the years a number of tools have been developed to deal with purely continuous or purely discrete systems. None of these tools, however, is capable of fully addressing the issues arising in hybrid systems. The development of specialized hybrid system tools has recently attracted the attention of a number of research groups [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. In this paper we summarize one such effort [18, 19, 20, 21].

In this paper, we concentrate on the problem of designing hierarchical, hybrid controllers for large scale systems. We start by giving an overview of the proposed design process in Section 2. The steps of the process are outlined and motivation is given for the detailed derivations that follow. In Section 3 we introduce a modeling formalism that allows us to model multiagent systems with hybrid dynamics in a modular fashion. Based on such models a more detailed algorithm for part of the design process is proposed in Section 4. The algorithm makes use of ideas from game theory to capture the interaction between the agents, derive continuous controllers and quantify the discrete coordination needed to achieve certain performance objectives. In Section 5, the application of this algorithm to the problem of automated vehicle following is presented. Section 6 gives a summary of the proposed scheme and highlights the major points. For further examples on the applications of these techniques the reader is referred to [20, 22].

2 Outline of the Design Process

In this section we give an outline of a possible strategy for designing hierarchical controllers for large scale systems. It involves a number of steps. For most of these steps we will only give an intuitive discussion of the issues involved and how one can proceed to resolve them. A more formal analysis of the remaining steps is given in Section 4. The design process can be split into two phases, a top–down phase where the problem is parsed to a preliminary hierarchy and the design specifications are quantified, and a bottom–up phase where the details of the hierarchy are filled in and the resulting closed loop performance is extracted.

2.1 Top–Down Phase

We assume that the starting point for the design is a model of the physical process (given in the modeling formalism of Section 3) and a description of the desired emergent behavior of the closed loop system. In most examples the desired behavior is given linguistically; for example for an automated highway problem the desired emergent behavior could be to achieve a higher throughput of the highway while maintaining the same level of safety and passenger comfort. The first step is to quantify such a specification. This process is difficult to formalize in an abstract setting. The solutions given in applications are typically problem dependent and rely heavily on the designers intuition about the problem. The reader is referred to [23] for an outline of this process for the automated highway problem and to [4] for the air traffic management problem.

The difficulties associated with quantifying an emergent behavior specification are beyond the scope of this paper. We circumvent them by assuming that the process has already been completed. Following the results of [4, 23] we assume that, at the end of the top down phase we are presented with a preliminary discrete design, in the form of a set of way points and a set of logic rules for switching between them. Such a design can be encoded by a finite state machine (see for example the design of [24] for the automated highways). The discrete design is accompanied by a set of cost functions to be used in the continuous controller design. A number of cost functions may need to be considered to encode different requirements (for example safety, efficiency, etc.). It will be assumed that the cost functions can be ranked in terms of importance (for example safety will be more important than efficiency). Acceptable performance can be encoded by means of thresholds on the final costs.

2.2 Bottom–Up Phase

The task facing the designer now is to derive continuous control laws and refinements on the preliminary discrete design to guarantee the specifications encoded by the cost functions and the thresholds. This process is the main focus of this paper.

2.2.1 Continuous Design

In terms of the continuous controllers we can distinguish two factors that influence the system evolution: the *control* that the designer is called upon to specify and the *disturbances* that enter the system, over which the designer has no control. We distinguish three classes of disturbances:

- **Class 1:** Exogenous signals, such as unmodeled forces and torques in mechanical systems, sensor noise, etc.
- **Class 2:** Unmodeled dynamics.
- **Class 3:** The actions of other agents, in a multiagent setting.

Disturbances of Class 1 and 2 are standard in classical control theory. Class 3 will be the most interesting one from the point of view of hybrid control. Recall that at this stage we are merely modeling the plant, therefore we assume no cooperation between the agents. As a result, each agent views the actions of its neighbors as uncontrollable disturbances.

The first objective is to derive a continuous design for the control inputs that guarantees performance despite the disturbances. The design of the continuous laws should be optimal with respect to the closed loop system requirements. A good tool for this kind of set up is game theory. In the game theoretic framework the control and the disturbances are viewed as adversaries in a game. The control seeks to improve system performance while the disturbance seeks to make it worse. Games like these do not necessarily have winners. If, however, we set thresholds on the cost functions to distinguish acceptable from unacceptable performance we can say that the control wins the game if the requirements are satisfied for any allowable disturbance, while the disturbance wins otherwise. The principles involved in game theoretic design are very similar to the ones for optimal control. Roughly speaking, the designer has to find the best possible control and the worst possible disturbance. If the

requirements are met for this pair, it is possible to obtain a satisfactory design (one such design is the "best possible" control). If the requirements are not satisfied the problem can not be solved as is, since there exists a choice of disturbance for which, no matter what the controller does, the closed loop system will fail to satisfy the requirements.

Game theoretic ideas have already been applied in this context to problems with disturbances of Class 1 and 2 and quadratic cost functions. The resulting controllers are the so called H_∞ or L_2 optimal controllers (see for example [25, 26]). We will try to extend these ideas to the multiagent, hybrid setting and focus on disturbances of Class 3. The process is described in Section 4.

2.2.2 Discrete Design

To complete the hierarchical controller the preliminary discrete design needs to be augmented to account for the performance of the continuous design. The solution to the game theoretic problem will produce continuous control laws and sets of initial conditions for which performance is guaranteed for any disturbance. If the preliminary discrete design dictates switching between control laws, these sets of initial conditions can be used as guidelines for the switching (a form of interface between the continuous and discrete domains).

If it turns out that the disturbance is such that the specifications can not be met for any controller the design fails. The only way to salvage the situation is to somehow limit the disturbance. For disturbances of Class 3 this may be possible by means of communication and coordination between the agents. The objective is to come up with a discrete design that limits the disturbance so that a continuous design is feasible. Unfortunately, no formal technique exists for determining the coordination necessary to solve such a problem. Our work on specific applications [20, 22] suggests that all the requirements imposed on the discrete design by the continuous controllers can be encoded in terms of timed language specifications. The work of [27] indicates that timed language problems are purely discrete. Therefore, in principle, the task of completing the hierarchical controller is tractable and can be tackled using standard computational discrete design tools.

In practice the augmentation of the discrete design is carried out based on heuristics and the insight gained from the continuous level design. The system is then analyzed to determine whether the coordination provides enough reduction in the disturbance for the game discussed above have a solution. In this paper we will limit our attention to the design of continuous control laws and interfaces between these laws and the discrete world. In the examples we consider (refer to [20, 22]) the requirements on the discrete design follow automatically.

Summarizing the bottom–up phase, our approach can be thought of as an attempt to add a minimal amount of centralization to an initial decentralized design. The plan is to locally optimize the system performance and establish bounds on the global performance under a the resulting control laws. Then we investigate how cooperation can be used to improve on these bounds. If a limit is set on certain aspects of the performance our analysis will allow us to determine the minimum amount of coordination needed to achieve those limits. This approach may be valuable for applications other than controller design, for example for distributing processes in parallel computers (so that the minimum amount of communication is needed) and for signal or image transmission (to determine the minimum

amount of information that an intelligent receiver needs to reproduce the transmitted signal).

2.2.3 Extracting the Emergent Behavior

To complete the bottom–up phase we need to somehow extract the emergent behavior produced by our design. This will allow us, for example, to compare various designs proposed for the same problem.

This step is strongly linked to the quantification of emergent behavior attempted at the top–down phase of the design. In the setting considered here, where the requirements on the continuous design are encoded by cost functions, a natural way of abstracting the emergent behavior is by using optimal control. For example, optimal control ideas can be used to obtain the minimum and maximum times that a hybrid system spends in each discrete state. These bounds can then be used as a rudimentary timed abstraction of the hybrid system. Similarly, the sets of initial conditions for which the performance requirements are satisfied, that are generated by solving the game theory problems, can be used as a different discrete abstraction ("safe" vs "unsafe" states for example).

The requirement for designer input limits the complexity of the problems that can be handled analytically using this approach. A more common approach is to investigate the system performance using *macro-simulation*. This approach is especially valuable for multi-agent problems and has been successfully applied to a number of examples [28, 29].

2.3 Multiagent Design and Verification Environment

Based on the discussion presented above, we envision developing a complete design, simulation and verification environment for multi-agent, hierarchical, hybrid control systems (Figure 1). The specifications are described by the desired emergent behavior of the collection of agents. These are simple requirements usually described linguistically: increased throughput and safety and reduced emissions for Automated Highway Systems, increased frequency of landings and takeoffs and optimum utilization of air space for Air Traffic Management Systems, etc.

These requirements have to get parsed into a system architecture. Designing a control architecture involves decomposing the system into a subsystem hierarchy, specifying the subsystem interconnections and determining the limits of the environmental inputs. We will concentrate on partially decentralized architectures, with coordinating semi-autonomous agent operation. The controller of each agent is described by a multi layer hierarchy. The control laws and the inter-agent coordination schemes are to be designed using design tools, in order to satisfy the specified properties such as safety, efficient resource utilization, etc. The design tools may be conventional discrete and continuous tools as well as specialized tools for hybrid control (such as the ones discussed in this paper). Once the control laws for individual agents are designed, we would like to be able to abstract their detailed behavior so that the collective emergent characteristics can be evaluated using tools such as macro-simulation.

The process can be customized to a specific application, such as Automated Highway Systems (AHS) or Air Traffic Management Systems (ATMS). As an example, consider the case of AHS. The hybrid system description language introduced above can be customized to the specific application domain as shown in [30]. Then controllers for the various maneuvers required on the AHS can be designed to satisfy safety, comfort and efficiency requirements

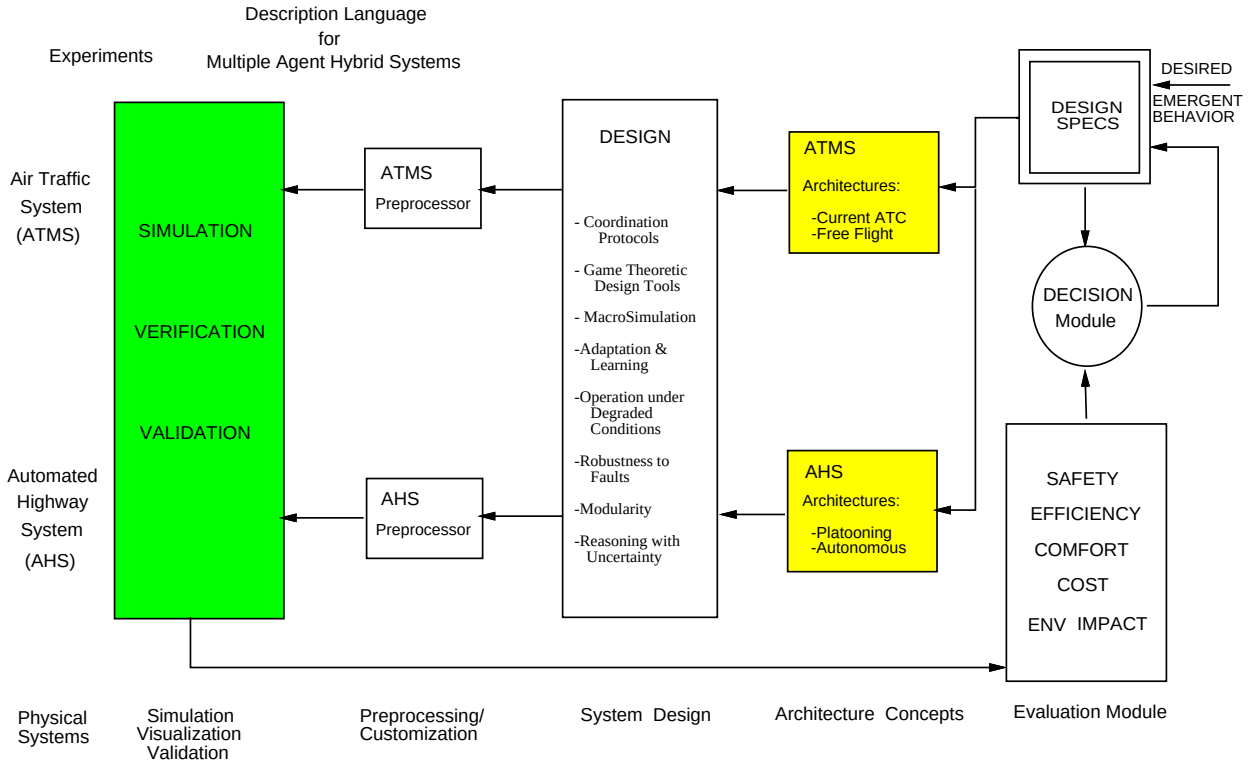

Figure 1: Multi-Agent Hybrid System Design Environment

using the tools presented in Section 4 [20]. To get to the emergent behavior, the various maneuver control laws of an automated vehicle can then be abstracted into space-time requirements. This information can be used in a traffic flow model to evaluate the throughput of the highway. The detailed designs can also be converted into a generic multi-agent hybrid system simulation language and simulated using a micro-simulator. One such micro-simulator SmartAHS is being built for AHS applications at Berkeley. Note that the simulation program should be able to handle dynamically changing inter-connections between different agents. A language for describing such a dynamically changing network of hybrid systems, called SHIFT, is described in [31]. SHIFT adds the functionality to dynamically create and destroy new components and change input output connections as part of the reset actions. Specifications in SHIFT can be verified or validated using hardware-in-the-loop experiments.

2.4 Related Problems: Verification using Optimal Control

Optimal control and gaming ideas may also prove useful for verification. Standard automatic verification techniques involve some form of exhaustive search, to verify that all possible runs of the system satisfy a certain property. An optimal control approach to verification, on the other hand, involves solving an optimal control problem to obtain the worst possible run with respect to a given requirement and then verifying that the specification holds for this run. If this is the case, it will also hold for all other runs. The main advantage is that, by removing the requirement for an exhaustive search, the limitations of **undecidability** and **computational**

complexity disappear. To guarantee that an automatic verification algorithm will terminate in a finite number of steps, the system needs to satisfy very stringent technical requirements. It can be shown [16] that relaxing any of these requirements makes the problem undecidable. Moreover, even though efficient algorithms exist (that make use of formal reduction techniques as well as heuristics and user input to facilitate the search), the verification task may still be prohibitively complex for current computers [32].

Verification of closed loop hybrid systems is better suited for optimal control, rather than game theory, as one of the two players (the controller) has its strategy fixed a-priori. Therefore only the disturbances, trying to do their worst to upset the design, enter the picture. An interesting class of disturbances that needs to be considered in this context is:

- **Class 4:** Commands from the discrete controller

From the point of view of the continuous system (where the optimal control problem is to be solved) these commands can be viewed as signals that make the continuous system switch between control laws, fixed points etc. Optimal control can be used to determine the discrete command sequences that force the continuous system to violate the performance specifications. If the discrete design is such that these command sequences are excluded then the hybrid design is verified. [33] discusses the application of these ideas to the automated highway example.

In this paper we will concentrate on controller design and will not discuss the problem of verification further. A more thorough presentation can be found in [19, 21].

3 Hybrid System Modeling

In this section, we present a rather general model for multiagent systems. Each agent is modeled as a hybrid dynamical system. The modeling framework is modular, in that a hybrid system can be specified as a composition of subsystems. Each agent specification is a tree structured hierarchy of subsystems, representing the plant dynamics, sensors, actuators, communication devices and controllers.

3.1 Hybrid Dynamical Systems

The basic entity of our models will be the **hybrid dynamical system** or **hybrid automaton** (the terms will be used interchangeably). Hybrid automata are convenient abstractions of systems with phased operation and they appear extensively in the literature in various forms ([11, 14]). The model we consider will be similar to models used primarily in computer science (in particular the ones in [14] and [15]). However, because we are interested in modeling different agents and their interaction we will take a more input/output approach, along the lines of the reactive module paradigm [34]. For an excellent overview of hybrid models from the dynamical systems point of view see [13].

Variables

A hybrid automaton is a dynamical system which determines the evolution and interaction of a finite collection of variables. We consider two distinct kinds of variables, **discrete** and **continuous**.

Definition 1 *A variable is called* **discrete** *if it takes values in a countable set and it is called* **continuous** *otherwise.*

We will assume no special algebraic structure for the values of the discrete variables. The only operations we will allow are assigning a value to a variable and checking whether the value of a variable and a member of the value set (or the values of two variables that take values in the same set) are equal. We assume that continuous variables take values in subsets of $\mathbb{R}^n$ for some value of n[1].

The variables in our model will be split into three classes: **Inputs** (external), **Outputs** (interface) and **State** (private)[2]. We will denote the input space (set where the input variables take values) by:

$$U = U_D \times U_C$$

the output space by:

$$Y = Y_D \times Y_C$$

and the state space by:

$$X = X_D \times X_C$$

The subscripts D and C indicate whether the variable is discrete or continuous. To avoid unnecessary subscripts we denote an element of U by u, an element of Y by y and an element of X by (q, x).

Time

Let T denote the set of times of interest. Our models will evolve in continuous time, so we will assume that $T = [t_i, t_f] \subset \mathbb{R}$. The definitions should easily extend to other sets T with appropriate topological and algebraic structure. The variables will evolve either continuously as a function of time or in instantaneous jumps. Therefore the evolution of the system will be over sets of the form:

$$\mathcal{T} = \{[\tau_0', \tau_1][\tau_1', \tau_2], \ldots [\tau_{n-1}', \tau_n]\} \tag{1}$$

with $\tau_i \in T$ for all i, $\tau_0' = t_i, \tau_n = t_f$ and $\tau_i = \tau_i' \leq \tau_{i+1}$ for all $i = 1, 2, \ldots, n-1$. The implication is that τ_i are the times where discrete jumps of the state or input occur. We will use τ to denote an element of $\mathcal{T}$.

Dynamics

The evolution of the variables will be determined by four objects:

$$I \subset X \tag{2}$$

$$f : X \times U \longrightarrow TX_C \tag{3}$$

$$E \subset X \times U \times X \tag{4}$$

$$h : X \times U \longrightarrow Y \tag{5}$$

[1]The definitions generalize to weaker conditions, for example continuous variables lying on manifolds.

[2]The terms in bold come from the system theory literature while the terms in brackets are more computer science oriented. They can be used interchangeably, though we stick to the terms in bold most of the time.

Here TX_C represents the tangent space of the space X_C. I is the set of possible values of the initial states, i.e.:

$$(q(\tau_0'), x(\tau_0')) \in I$$

The vector field, f, determines the evolution of the continuous state on intervals of the form $[\tau_{i-1}', \tau_i]$ with $\tau_{i-1}' < \tau_i$. For every $t \in [\tau_{i-1}', \tau_i]$:

$$\dot{x}(t) = f(q(t), x(t), u(t))$$

Without loss of generality we will assume that f is time invariant[3]. We make the standard assumptions on f for existence and uniqueness of solutions to the ordinary differential equation.

The set E determines the discrete evolution of the state. The interpretation is that the state can keep evolving continuously as long as:

$$(q(t), x(t), u(t), q(t), x(t)) \in E$$

The state can take a discrete jump at time τ_i from $(q(\tau_i), x(\tau_i)) \in X$ to $(q(\tau_i'), x(\tau_i')) \in X$ if:

$$(q(\tau_i), x(\tau_i), u(\tau_i), q(\tau_i'), x(\tau_i')) \in E$$

Finally, h determines the output evolution. For all $t \in \tau$:

$$y(t) = h(q(t), x(t), u(t))$$

Collecting the above elements we give the following definitions:

Definition 2 *A* **hybrid dynamical system**, *H, is a collection (X, U, Y, I, f, E, h), with:*

$$\begin{aligned} X &= X_D \times X_C \\ U &= U_D \times U_C \\ Y &= Y_D \times Y_C \\ I &\subset X \\ f &: X \times U \longrightarrow TX_C \\ E &\subset X \times U \times X \\ h &: X \times U \longrightarrow Y \end{aligned}$$

where X_C, U_C, Y_C are respectively open subsets of $\mathbb{R}^n, \mathbb{R}^m, \mathbb{R}^p$, for some finite values of n, m, p and X_D, U_D, Y_D are countable sets.

Definition 3 *A* **run** *of the hybrid dynamical system H over an interval $T = [t_i, t_f]$ consists of a collection (τ, q, x, y, u) with $\tau \in \mathcal{T}$, $q : \tau \to X_D$, $x : \tau \to X_C$, $y : \tau \to Y$ and $u : \tau \to U$ which satisfies the following properties:*

[3]With some additional notation overhead the same definition can be given in terms of the flow of the vector field. The advantage to this would be that the definition would directly extend to other cases, such as discrete time systems.

1. **Initial Condition**: $(q(\tau_0'), x(\tau_0')) \in I$.

2. **Discrete Evolution**: $(q(\tau_i), x(\tau_i), u(\tau_i), q(\tau_i'), x(\tau_i')) \in E$, *for all* i.

3. **Continuous Evolution**: *for all* i *with* $\tau_i' < \tau_{i+1}$ *and for all* $t \in [\tau_i', \tau_{i+1}]$:

$$\dot{x}(t) = f(q(t), x(t), u(t)) \tag{6}$$
$$q(t) = q(\tau_i') \tag{7}$$
$$(q(t), x(t), u(t), q(t), x(t)) \in E \tag{8}$$

4. **Output Evolution**: *for all* $t \in \tau$

$$y(t) = h(q(t), x(t), u(t))$$

It can be shown [21] that the definitions introduced here are rich enough to allow us to model regular dynamical systems, discrete events, autonomous jumps, controlled jumps, etc.

3.2 Graphical Representation

If the discrete state can assume a finite number of values it is very convenient to represent the hybrid automaton by a finite graph. We can associate a finite graph to a given hybrid automaton H using the following construction.

Construction 1 (Graph Generation):

Nodes: the number of nodes in the graph will be equal to the number of possible values of the discrete state. The nodes will be indexed by a corresponding discrete state value, $q \in X_D$.
Continuous Evolution: to each node, q, we associate a vector field, f_q:

$$\begin{aligned} f_q : X_C \times U &\longrightarrow TX_C \\ (x, u) &\longmapsto f(q, x, u) \end{aligned} \tag{9}$$

The implication is that while in the node q the continuous state evolves according to f_q.
Node Invariants: To each node, q, we associate an invariant:

$$\mathrm{Inv}_q = \bigcup\{(x, u) | x \in X_C, u \in U, (q, x, u, q, x) \in E\} \subset X_C \times U \tag{10}$$

The interpretation is that the system can remain in node q if and only if $(x, u) \in \mathrm{Inv}_q$.
Transition Guards: To the transition from node q to node q' we associate a guard:

$$\mathrm{En}_{qq'} = \bigcup\{(x, u) | x, x' \in X_C, u \in U, (q, x, u, q', x') \in E\} \subset X_C \times U \tag{11}$$

The interpretation is that the transition can take place if and only if $(x, u) \in \mathrm{En}_{qq'}$.
Transition Reset: To the transition from node q to node q' we associate a set valued map:

$$\mathrm{Res}_{qq'}(x, u) = \bigcup\{x' | x' \in X_C, (q, x, u, q', x') \in E\} \subset X_C \tag{12}$$

The interpretation is that if the transition takes place from (x, u) then after the transition the state finds itself in (q', x') with $x' \in \mathrm{Res}_{qq'}(x, u)$. □

The above construction allows us to represent a hybrid automaton graphically as shown in Figure 2. Note that there is no requirement that $q \neq q'$, i.e. loops to the same node are allowed. In ensuing sections some of the elements of the graph may be omitted to simplify the figures. The interpretation will be that a missing invariant or guard is equal to $X_C \times U$ while a missing reset map is the identity in x for all $u \in U$.

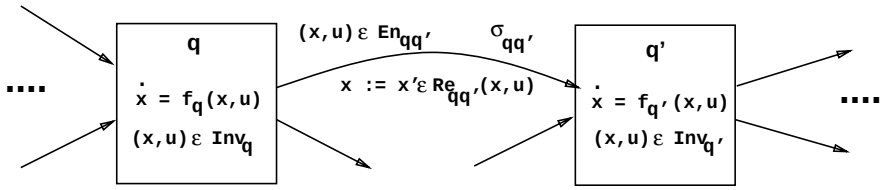

Figure 2: Hybrid automaton graph nodes

3.3 Operations on Hybrid Dynamical Systems

We will only define three operations on hybrid dynamical systems: interconnection, renaming and hiding. Interconnection will allow us to form new hybrid systems out of collections of existing ones, renaming will allow us to connect systems with their subsystems while hiding will hide some outputs of a given hybrid system from the rest of the world.

Let $\{H_i\}_{i=1}^N$ be a collection of hybrid automata, $H_i = \{X_i, U_i, Y_i, I_i, f_i, E_i, h_i\}$. We can write the inputs and outputs in vector form as:

$$u_i = \begin{bmatrix} u_{i,1} \\ \vdots \\ u_{i,m_i} \end{bmatrix} \in U_i \quad y_i = \begin{bmatrix} y_{i,1} \\ \vdots \\ y_{i,p_i} \end{bmatrix} \in Y_i$$

Let:

$$\begin{aligned} \hat{U} &= \{(1,1),(1,2),\ldots,(1,m_1),(2,1),\ldots,(2,m_2)\ldots,(N,1),\ldots,(N,m_N)\} \\ \hat{Y} &= \{(1,1),(1,2),\ldots,(1,p_1),(2,1),\ldots,(2,p_2),\ldots,(N,1),\ldots,(N,p_N)\} \end{aligned}$$

Definition 4 *An* **interconnection**, $\mathcal{I}$, *of a collection of finite automata,* $\{H_i\}_{i=1}^N$, *is a partial map:*

$$\mathcal{I} : \hat{U} \longrightarrow \hat{Y}$$

An interconnection of hybrid automata can be though of as a pairing $(u_{i,j}, y_{k,l})$ of inputs and outputs. An interconnection is only a partial map (i.e. some inputs may be left free), need not be surjective (i.e. some outputs may be left free) and need not be injective (i.e. an output may be paired with more than one input). Let $Pre(\mathcal{I})$ be the subset of $\hat{U}$ for which the partial map $\mathcal{I}$ is defined. Also let Π_α denote the projection of a vector valued quantity to the element with index α.

Definition 5 *Given a collection of hybrid automata* $\{H_i\}_1^N$ *and an interconnection* $\mathcal{I}$, *the symbolic operation* **substitution**, *denoted by* $\rightsquigarrow$, *assigns to each input,* $u_{i,j}$ *a map on* $X_1 \times \ldots \times X_N \times U_1 \times \ldots \times U_N$, *according to:*

$$u_{i,j} \rightsquigarrow \begin{cases} u_{i,j} & \text{if } (i,j) \notin Pre(\mathcal{I}) \\ h_{\mathcal{I}(i,j)} : X_{\Pi_1(\mathcal{I}(i,j))} \times U_{\Pi_1(\mathcal{I}(i,j))} \to Y_{\Pi_1(\mathcal{I}(i,j))} & \text{if } (i,j) \in Pre(\mathcal{I}) \end{cases}$$

If for all $(i,j) \in Pre(\mathcal{I})$, $Y_{\mathcal{I}(i,j)} \subset U_{i,j}$, *operation* $\rightsquigarrow$ *can be repeatedly applied to the right hand side by appropriate map compositions. The construction terminates for each* $u_{i,j}$ *if the right hand side either contains* $u_{i,j}$ *itself or contains only* $u_{k,l} \notin Pre(\mathcal{I})$. *The resulting map will be denoted by* $(u_{i,j} \rightsquigarrow^*)$.

Because there are a finite number of inputs, the construction of $(u_{i,j} \rightsquigarrow^*)$ terminates in a finite number of steps.

To ensure that an interconnection is well defined as an operation between hybrid automata we impose the following technical conditions:

Definition 6 *An interconnection, $\mathcal{I}$, of a collection of hybrid dynamical systems, $\{H_i\}_{i=1}^N$, is* **well posed** *if:*

1. *For all $(i,j) \in Pre(\mathcal{I})$, $Y_{\mathcal{I}(i,j)} \subset U_{i,j}$,*
2. *There are no algebraic loops, i.e. for all $(i,j) \in Pre(\mathcal{I})$ the map $(u_{i,j} \rightsquigarrow^*)$ does not involve $u_{i,j}$.*

These requirements imply that $(u_{i,j} \rightsquigarrow^*)$ is well defined as a map between the following spaces:

$$(u_{i,j} \rightsquigarrow^*) : X_1 \times \ldots \times X_N \times \Pi_{\hat{U} \setminus Pre(\mathcal{I})}(U_1 \times \ldots \times U_N) \longrightarrow \Pi_{i,j}(U_1 \times \ldots \times U_N)$$

Fact 1 *Every well posed interconnection, $\mathcal{I}$, of a collection of hybrid dynamical systems, $\{H_i\}_{i=1}^N$, defines a new hybrid dynamical system.*

Proof: Let $H = \{X, U, Y, I, f, E, h\}$ denote the interconnection automaton, defined by $X = X_1 \times \ldots \times X_N$, $U = \Pi_{\hat{U} \setminus Pre(\mathcal{I})}(U_1 \times \ldots \times U_N)$, $Y = Y_1 \times \ldots \times Y_N$, $I = I_1 \times \ldots \times I_N$, $f = [f_i \circ (u_i \rightsquigarrow^*)]_{i=1}^N$, $E \subset X \times U \times X$ with $e = (q, x, u, q', x') \in E$ if and only if for all $i = 1, \ldots, N$:

$$(q_i, x_i, (u_i \rightsquigarrow^*)(q, x, u), q_i', x_i') \in E_i$$

and $h = [h_i \circ (u_i \rightsquigarrow^*)]_{i=1}^N$. $\square$

The expression $(u_i \rightsquigarrow^*)$ denotes the map generated by applying $\rightsquigarrow^*$ to the elements $u_{i,1}, \ldots, u_{i,m_i}$. The terms in square brackets have the obvious interpretation as vectors. The symbol $\circ$ denotes composition of maps. By a slight abuse of notation, we will refer to H itself as the interconnection of $\{H_i\}_{i=1}^N$.

Definition 7 *Consider a hybrid automaton H with input space $U = U_1 \times \ldots \times U_m$ and output space $Y = Y_1 \times \ldots \times Y_p$ and variables $\hat{u}_i \in \hat{U}_i \subset U_i$ and $\hat{y}_j \in \hat{Y}_j \supset Y_j$.* $\mathbf{Rename}_{u_i \to \hat{u}_i}(H)$ *is a new automaton with the same state and output spaces, input space:*

$$\hat{U} = U_1 \times \ldots U_{i-1} \times \hat{U}_i \times U_{i+1} \times \ldots \times U_m$$

the same initial condition set and $\hat{f}, \hat{E}$ and $\hat{h}$ equal to the restrictions of the corresponding quantities of H to $\hat{U}$. Similarly, $\mathbf{Rename}_{y_i \to \hat{y}_i}(H)$ *is a new automaton with the same state and input spaces, output space:*

$$\hat{Y} = Y_1 \times \ldots Y_{j-1} \times \hat{Y}_j \times Y_{j+1} \times \ldots \times Y_p$$

and the same dynamics.

This operation can be used when a hierarchy of hybrid automata is constructed to identify inputs and outputs of systems and their subsystems.

Definition 8 *Given a hybrid automaton H with output space $Y = Y_1 \times \ldots \times Y_p$ and an output y_i taking values in Y_i the operation* $\mathbf{Hide}_{y_i}(H)$ *produces a new hybrid automaton, with the same state and input spaces, output space:*

$$Y_1 \times \ldots \times Y_{i-1} \times Y_{i+1} \times \ldots Y_p$$

the same dynamics as H and output map $[h_1 \ \ldots \ h_{i-1} \ h_{i+1} \ \ldots \ h_m]^T$.

This operation can be used when hybrid automata are interconnected to form agents to hide certain local features of an agent from the rest of the world.

3.4 Agent Model

Each agent will be modeled as an interconnection of hybrid dynamical systems. The intuition is that each subsystem will be used to describe a distinct functionality of the agent. Typically an agent will contain subsystems modeling the plant dynamics, the sensors, the actuators, the continuous and discrete controllers and the communication devices.

For the purpose of modeling multiagent systems we will distinguish three kinds of inputs for each subsystem:

1. **Control inputs**, that can be specified locally (i.e. within the agent) by interconnections to the outputs of other subsystems. They reflect the actions that the agent may decide to take.

2. **Environmental inputs** (or **disturbances**), that can not be locally specified and reflect actions of the environment (such as sensor noise or the effect of unmodeled dynamics) or actions of other agents.

3. **Coordination inputs**, that are used for interagent cooperation, for example through communication protocols.

This classification of inputs is motivated by our approach to the control of multiagent systems which is based on semiautonomous agent operation.

The agent model will itself be a dynamical system. Within it, it may contain an entire hierarchy of interconnected subsystems. The agent inputs and outputs will be connected to the subsystem inputs and outputs according to a set of rules:

- system input and subsystem input
- system output and subsystem output
- subsystem input and subsystem output

The coupling between subsystems can be implemented with the *interconnection* operation. The coupling between system and subsystem inputs and outputs can be implemented by *renaming*. It is assumed that any redundant subsystem outputs are *hidden* once the agent is formed.

The dynamic evolution and the subsystem interconnection rules are defined so that the agent model can be "flattened" into a single equivalent hybrid dynamical system. As

before, let $(q, x) \in X$ denote the state of the agent (discrete and continuous). Also let $u \in U_D \times U_C$ denote the control inputs and $d \in D_D \times D_C$ denote the environmental inputs. Here we will concentrate more on the design of the low level controllers, therefore we will not use special symbols for the coordination inputs.

The dynamics of the flattened agent model over a set of times of interest $T = [t_i, t_f]$ will be determined by a set of relations of the form:

Initial condition	$(q(t_i), x(t_i)) = (q^0, x^0) \in I$
Continuous evolution	$\begin{cases} \dot{x}(t) = f(q(t), x(t), u(t), d(t)) \\ (q(t), x(t), u(t), d(t), q(t), x(t)) \in E \end{cases}$
Discrete evolution	$(q, x, u, d, q', x') \in E$ for jumps $(q, x) \rightarrow (q', x')$
Output evolution	$y(t) = h(x(t), u(t), d(t))$

Physical considerations (such as actuator saturation) impose certain restrictions on the system evolution. We encode these restrictions in terms of constraints on the state, input and disturbance trajectories:

$$(q(), x()) \in \mathcal{Q} \times \mathcal{X} \subset PC(\mathcal{T}, X_D) \times PC^1(\mathcal{T}, X_C) \quad (13)$$

$$u() \in \mathcal{U} \subset PC(\mathcal{T}, U) \quad (14)$$

$$d() \in \mathcal{D} \subset PC(\mathcal{T}, D) \quad (15)$$

$PC(\mathcal{T}, *)$ denotes the set of piecewise continuous maps from $\mathcal{T}$ to $*$, whereas $PC^1(\mathcal{T}, *)$ the set of piecewise differentiable maps[4]. We will use the symbols PC and PC^1 to denote these sets whenever there is no ambiguity about the domain and range.

Of particular interest for continuous variables are constraints that can be encoded by requiring that the variable lies in a certain set for all times, i.e. for all $t \in \tau$:

$$\begin{aligned} x(t) &\in \mathbf{X} \subset \mathbb{R}^n \\ u(t) &\in \mathbf{U} \subset \mathbb{R}^m \\ d(t) &\in \mathbf{D} \subset \mathbb{R}^p \end{aligned}$$

It should be noted that this class of constraints excludes certain important cases such as non-holonomic and "isoperimetric" constraints.

For discrete variables pure "value" constraints play less of a role. A discrete variable can be though of as a piecewise constant function of time (multi valued at the transition points). Two aspects of such a function are of interest:

- The order in which the values are observed.
- The times at which the function jumps from one value to the next.

The constraint set will impose limits on these two aspects of the evolution of the discrete variables. It should be noted that both these constraints can be encoded by the requirement that the discrete variable sequence is generated/accepted by a timed automaton [27].

[4]The definitions are straightforward generalizations of the corresponding definitions for $T = [t_i, t_f]$.

3.5 Mathematical Tools

The analysis presented in the next section will require concepts from the areas of game theory, optimal control, dynamical systems and topology. Here we briefly introduce only the necessary game theoretic notions. We will be dealing the following type of two player, zero sum games:

Definition 9 *A* **two player, zero sum dynamic game** *involves:*

1. *A time interval* $[0, T]$.
2. *A trajectory space,* $\mathcal{X}$ *with some topological structure. The elements of* $\mathcal{X}$*, denoted* $\{x(t), 0 \leq t \leq T\}$*, are the permissible state trajectories of the game.*
3. *An input space,* $\mathcal{U}$*, with some topological structure. The elements of* $\mathcal{U}$*, denoted* $\{u(t), 0 \leq t \leq T\}$*, are the permissible inputs of player 1.*
4. *An disturbance space,* $\mathcal{D}$*, with some topological structure. The elements of* $\mathcal{D}$*, denoted* $\{d(t), 0 \leq t \leq T\}$*, are the permissible inputs of player 2.*
5. *A differential equation:*

$$\dot{x}(t) = f(t, x(t), u(t), d(t)) \tag{16}$$
$$x(0) = x^0 \tag{17}$$

   whose solution determines the state trajectory for a given selection of u *and* d*.*

6. *A cost function:*

$$J : \mathcal{X} \times \mathcal{U} \times \mathcal{D} \longrightarrow \mathbb{R}^+ \tag{18}$$

To be consistent with the notation we develop for the multiagent models we will give the following interpretation: the "reward" of player 1 for a given play is $-J(x, u, d)$ while the reward of player 2 is $J(x, u, d)$ (hence a zero sum game). In other words, player 1 is trying to minimize J while player 2 is trying to maximize it.

For our games we will assume the so called **closed-loop, perfect state information structure**. This means that, when called upon to decide their strategy at time t, both players have access to the entire state $\{x(s), 0 \leq s \leq t\}$ up to that point. For our purposes we will assume that $x(t) \in X \subset \mathbb{R}^n$, $u(t) \in U \subset \mathbb{R}^{n_i}$ and $d(t) \in D \subset \mathbb{R}^{n_d}$ for all $t \in [0, T]$ and some n, n_i and n_d. We will also assume that $\mathcal{X}$ (respectively $\mathcal{U}$ and $\mathcal{D}$) will be a subset of the set of piecewise differentiable (respectively piecewise continuous) functions of t. Note that, if the differential equation (16) defines a unique state trajectory for a given choice of u and d, we can write the cost function as a function of the initial condition, rather than the whole state trajectory, i.e. $J : X \times \mathcal{U} \times \mathcal{D} \to \mathbb{R}$.

We will be interested in saddle solutions to these games:

Definition 10 *A* **saddle solution** *to the two player, zero sum game is a pair of input trajectories* $(u^*, d^*) \in \mathcal{U} \times \mathcal{D}$ *such that for any* $u \in \mathcal{U}$ *and any* $d \in \mathcal{D}$*:*

$$J(x^0, u^*, d) \leq J(x^0, u^*, d^*) \leq J(x^0, u, d^*)$$

In other words, a saddle solution will be such that any unilateral deviation from it will leave the player who decided to deviate worse off. The games considered in the examples of this dissertation will turn out to have unique saddle solutions. Existence and uniqueness of solutions can not be guaranteed for general games, however. For results in this direction the reader is referred to [35, 36, 37].

The saddle solution is not the only solution concept of interest in dynamic games. Other types of solution can be defined (Nash, Stackelberg, etc.). The solution concept will depend, among other things, on the nature of the game (discrete vs. continuous, deterministic vs. stochastic), the number of players and the information structure. The saddle solution used here is the simplest solution concept and applies only to two player, zero sum games. For a thorough treatment of dynamic games, solution concepts and applications the reader is referred to [36, 37, 38].

4 Game Theoretic Framework

In this section we give an algorithm to produce low level controllers for a hierarchical control scheme for large scale systems. As discussed in Section 2, the algorithm makes use of game theory to generate continuous controllers and consistent discrete abstractions for the resulting closed loop system.

4.1 Preliminary Discrete Design

We assume that the top–down phase of the design process has been completed. The discrete design can then be abstracted for the benefit of the lower levels in terms of three quantities:

- A sequence of desired way points y_j^d, $j = 1, 2, \ldots$, that should be tracked.
- For each way point, a set of design specifications (J_i, C_i), $i = 1, \ldots, N$. These are pairs of cost functions:
$$J_i : PC \times PC^1 \times PC \times PC \longrightarrow \mathbb{R} \tag{19}$$
mapping a run $(q(), x(), u(), d())$ of the agent automaton to $\mathbb{R}$, and thresholds $C_i \in \mathbb{R}$. An acceptable trajectory must be such that $J_i(q, x, u, d) \leq C_i$ for all $i = 1, \ldots, N$

We assume that the design specifications are ordered in the order of decreasing importance. Qualitatively, the most important cost functions encode performance aspects such as safety, while the least important ones encode performance aspects such as resource utilization. The design should be such that the most important requirements are not violated in favor of the less important ones, in other words the design should lead to $J_i \leq C_i$ whenever possible, even if this means that $J_j > C_j$ for some $j > i$.

The cost functions can "penalize" various aspects of the system runs. A typical cost function will involve a combination of the following elements:

1. Continuous evolution costs, i.e. costs associated with the evolution of the continuous state under the vector field f. These are the kinds of costs usually encountered in optimal control. A special case of particular interest is when each pair of inputs (u, d)

generates a unique state trajectory for a given initial condition (q^0, x^0). Then the cost function can be thought of as a map:

$$J_i : I \times PC \times PC \longrightarrow \mathbb{R} \tag{20}$$

When the discrete evolution of hybrid automaton may be nondeterministic, it may not be possible to obtain J_i in the form of equation (20).

2. Discrete evolution costs, i.e. costs associated with jumps of the states, inputs and disturbances from one value to another. For example, if a jump in the state takes place at time τ the cost associated with it may be thought of as a function:

$$J_i : X \times U \times D \times X \longrightarrow \mathbb{R}$$

that maps $(q(\tau), x(\tau), u(\tau), d(\tau), q(\tau'), x(\tau')) \in E$ to $\mathbb{R}$. Similar costs can be associated to input and disturbance jumps.

As discussed in Section 2, the question of how emergent behavior objectives, which are usually given linguistically, get parsed to way points, cost functions and thresholds is a topic for further research.

4.2 Continuous Layer

We present a technique for systematically constructing controllers which track the way points determined by the discrete layer and are optimal with respect to the given cost functions. In this section we restrict our attention to cost functions associated with the continuous state evolution and in particular functions of the form (20). This restricted class of functions suffices for all the examples considered so far in our work [20, 22]. We will also assume that the entire state of the plant is available for feedback.

4.2.1 Multiobjective Controller Design Algorithm

At the first stage we treat the design process as a two player, zero sum, dynamic game with cost J_1. One player, the control u, is trying to minimize the cost, while the other, the disturbance d, is trying to maximize it. Assume that the game has a saddle point solution, i.e. there exist input and disturbance trajectories, u_1^* and d_1^* such that:

$$\begin{aligned} J_1^*(q^0, x^0) &= \max_{d \in \mathcal{D}} \min_{u \in \mathcal{U}} J_1(q^0, x^0, u, d) \\ &= \min_{u \in \mathcal{U}} \max_{d \in \mathcal{D}} J_1(q^0, x^0, u, d) \\ &= J_1(q^0, x^0, u_1^*, d_1^*) \end{aligned}$$

The set:

$$V_1 = \{(q, x) \in X | J_1^*(q, x) \leq C_1\}$$

is the set of all initial conditions for which there exists a control such that the objective on J_1 is satisfied for the worst possible allowable disturbance (and hence for any allowable disturbance). u_1^* can now be used as a control law. It will guarantee that J_1 is minimized

for the worst possible disturbance. Moreover if the initial state is in V_1 it will also guarantee that the performance requirement on J_1 is satisfied. u_1^* however does not take into account the requirements on the remaining J_i's. To include them in the design let:

$$\mathcal{U}_1(q^0, x^0) = \{u \in \mathcal{U} | J_1(q^0, x^0, u, d_1^*) \leq C_1\} \tag{21}$$

Clearly:

$$\mathcal{U}_1(q^0, x^0) \begin{cases} = \emptyset & \text{for } (q^0, x^0) \notin V_1 \\ \neq \emptyset & \text{for } (q^0, x^0) \in V_1, \text{ as } u_1^* \in \mathcal{U}_1(q^0, x^0) \end{cases}$$

The set $\mathcal{U}_1(q^0, x^0)$ is the subset of admissible controls which guarantee that the requirement on J_1 is satisfied, whenever possible. Within this class of controls we would like to select the one that minimizes the cost function J_2. Again we pose the problem as a zero sum dynamic game between control and disturbance. Assume that a saddle solution exists, i.e. there exist u_2^* and d_2^* such that:

$$\begin{aligned} J_2^*(q^0, x^0) &= \max_{d \in \mathcal{D}} \min_{u \in \mathcal{U}_1(q^0, x^0)} J_2(q^0, x^0, u, d) \\ &= \min_{u \in \mathcal{U}_1(q^0, x^0)} \max_{d \in \mathcal{D}} J_2(q^0, x^0, u, d) \\ &= J_2(q^0, x^0, u_2^*, d_2^*) \end{aligned}$$

The set:

$$V_2 = \{(q, x) \in X | J_2^*(q, x) \leq C_2\}$$

contains the initial conditions for which there exists a control such that for any allowable disturbance the requirements on both J_1 and J_2 are satisfied. As the minimax problem can only be posed when $\mathcal{U}_1(q^0, x^0) \neq \emptyset$ we assume that $V_2 \subset V_1$. To introduce the remaining cost functions to the design we again define:

$$\mathcal{U}_2(q^0, x^0) = \{u \in \mathcal{U}_1(q^0, x^0) | J_2(q^0, x^0, u, d_2^*) \leq C_2\} \tag{22}$$

i.e. the subset of admissible controls that satisfy the requirements on both J_1 and J_2 for any disturbance.

The process can be repeated for the remaining cost functions. At the $i+1^{\text{st}}$ step we are given a set of admissible controls $\mathcal{U}_i(q^0, x^0)$ and a set of initial conditions V_i such that for all $(q^0, x^0) \in V_i$ there exists $u_i^* \in \mathcal{U}_i(q^0, x^0)$ that for all $d \in \mathcal{D}$ and for all $j = 1, \ldots, i$ leads to $J_j(q^0, x^0, u_i^*, d) \leq C_j$. Assume the two player, zero sum dynamic game for J_{i+1} has a saddle solution, u_{i+1}^*, d_{i+1}^*:

$$\begin{aligned} J_{i+1}^*(q^0, x^0) &= \max_{d \in \mathcal{D}} \min_{u \in \mathcal{U}_i(q^0, x^0)} J_{i+1}(q^0, x^0, u, d) \\ &= \min_{u \in \mathcal{U}_i(q^0, x^0)} \max_{d \in \mathcal{D}} J_{i+1}(q^0, x^0, u, d) \\ &= J_{i+1}(q^0, x^0, u_{i+1}^*, d_{i+1}^*) \end{aligned}$$

Define:

$$V_{i+1} = \{(q, x) \in X | J_{i+1}^*(q, x) \leq C_{i+1}\}$$

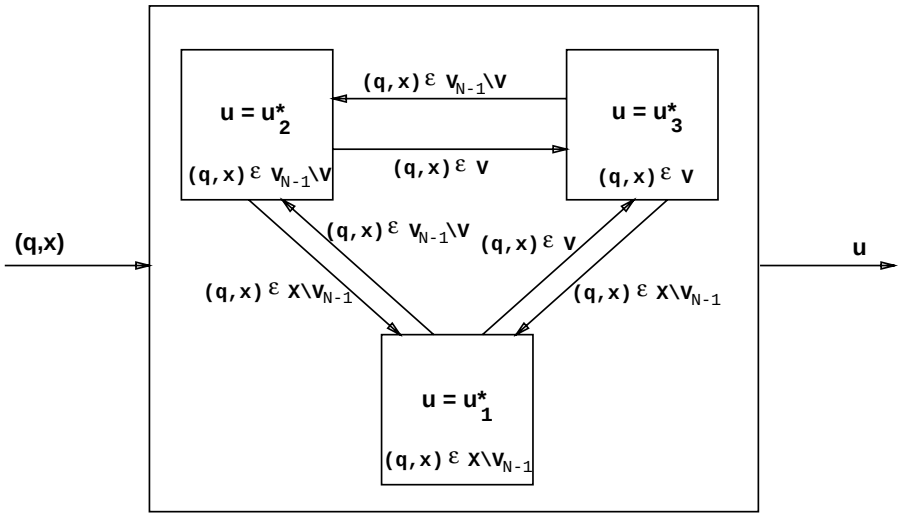

Figure 3: Controller automaton for switching between control objectives

and:

$$\mathcal{U}_{i+1}(q^0, x^0) = \{u \in \mathcal{U}_i(q^0, x^0) | J_{i+1}(q^0, x^0, u, d^*_{i+1}) \leq C_{i+1}\} \tag{23}$$

The process can be repeated until the last cost function. The result is a control law u^*_N and a set of initial conditions $V_N = V$ such that for all $(q^0, x^0) \in V$, for all $d \in \mathcal{D}$ and for all $j = 1, \ldots, N$, $J_j(x^0, u^*_N, d) \leq C_j$.

4.2.2 Controller Automaton

The controller can be extended to values of the state in the complement of V using the following switching scheme:

$$u^*(q,x) = \begin{cases} u^*_N(q,x) & (q,x) \in V \\ u^*_{N-1}(q,x) & (q,x) \in V_{N-1} \setminus V \\ \ldots & \ldots \\ u^*_1(q,x) & (q,x) \in X \setminus V_2 \end{cases} \tag{24}$$

Thus, even for a single set point, the resulting controller given by equation (24) involves switching due to multiple objective functions J_i. It therefore has to be implemented by a hybrid automaton. An example of such an automaton in a three cost function situation is shown in Figure 3. This procedure has to be repeated for *each set point* provided by the discrete layer.

4.2.3 Technical Issues

The above algorithm may run into technical difficulties, as there is no guarantee that the dynamic games will have a saddle solution, there is no straight-forward way of computing $\mathcal{U}_i(x^0)$ and there is no guarantee that the sets V_i (and consequently $\mathcal{U}_i(x^0)$) will be non-empty. Fortunately, in our examples thus far, including the most complicated cases of [20], a solution can be obtained analytically, or using simple numerical calculations. However, we can not

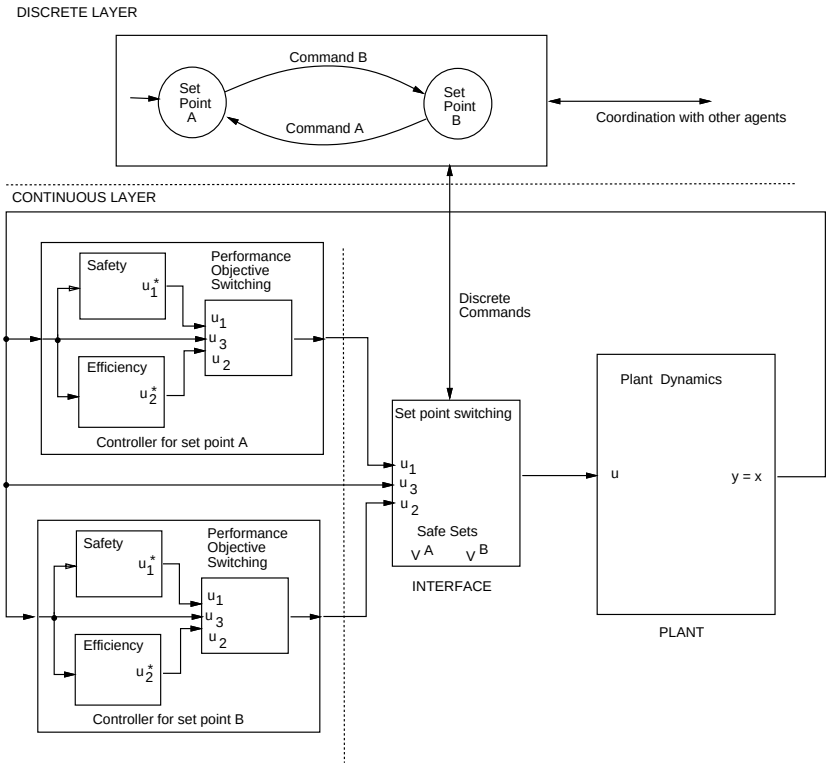

Figure 4: Hybrid controller for a single agent

expect such luck in general. New and sophisticated optimal control tools [39] will hopefully make the solution of more general problems feasible, at least numerically.

4.3 Interface and Discrete Design Revisited

The sets V are such that for all initial conditions in them all requirements on system performance are guaranteed. These sets impose conditions that the discrete switching scheme needs to satisfy. The discrete layer should not issue a new command (encoded by a way point) if the current state does not lie in the set V for the associated controller. Essentially, these sets offer a way of consistently abstracting performance of the continuous layer.

It should be noted that, by construction, the sets V_i are nested. Therefore there is a possibility that an initial condition lies in V_i for some $i = 1, \ldots, N' < N$ but not in V. This implies that certain requirements on the system performance (e.g. safety) can be satisfied, while others (e.g. efficient resource utilization) can not. This allows the discrete design some more freedom. For example, a new command may be issued if it is dictated by safety, even though it violates the requirements of efficiency. This construction provides a convenient way of modeling gradual performance degradation, where lower priority performance requirements are abandoned in favor of higher priority ones. It can be particularly useful for operation under degraded conditions, for example in the presence of faults [40].

The overall continuous design including the interface is shown in Figure 4. Switching of continuous controllers takes place at two levels. Assume that the discrete layer specifies two set points, A and B, and two objectives, for example safety and efficiency. For each set point,

the game theoretic framework will produce optimal controllers[5] for safety u_1^* and efficiency u_2^* along with the safe sets of initial conditions, V^A and V^B. As illustrated above, switching between these two controllers will be carried out depending on the current value of the state (represented in the figure as an input u_3 to the switching scheme). After receiving the new set point command from the discrete layer, the interface will switch to a new controller if the system state belongs to the corresponding safe set. If this requirement is not satisfied, and the discrete controller insists on the new way point, the discrete layer needs to coordinate with other agents. This type of coordination is what makes the operation of the agents semi-autonomous (as opposed to completely autonomous). It can be viewed as a way of restricting the domain, $\mathcal{D}$, of the disturbances. This "biases" the game in favor of the control input, hence enlarges the set of safe states and hopefully makes the transition to the new way point feasible. A more abstract view of the effect of coordination in this setting is as a way of turning the zero sum, non-cooperative game to a game with partial cooperation between the players. In extreme cases (for example presence of faults) a fully cooperative game may be needed to salvage the situation (as can be seen in the algorithms proposed in [40, 41]).

5 The Vehicle Following Example

To illustrate how this design methodology can be useful in applications we consider the problem of vehicle following on an automated highway. This is one of the problems that arise in the process of designing controllers for automated highways. Stronger results in this direction, in particular a complete hybrid controller that supports the formation of platoons with guaranteed safety, can be found in [20].

Consider three vehicles (labeled A, B and C) moving along a single lane highway (Figure 5). We will primarily be interested in the interaction between vehicles A and B, vehicle C will be used, to isolate the system A-B from the rest of the highway. Assume that vehicles A and B have lengths L_A and L_B and let x_A and x_B denote their positions with respect to a fixed reference on the road. Assume that vehicle B is leading while vehicle C comes last, i.e. $x_B > x_A > x_C > 0$. We assume no control over vehicle B and try to control vehicle A. The dynamics of the trailing vehicle be approximated [42] by a third order ordinary differential equation:

$$\dddot{x}_A = b_A(\dot{x}_A, \ddot{x}_A) + a_A(\dot{x}_A)v_A$$

a_A and b_A are complicated nonlinear functions of the state with $a_A(\dot{x}_A) \neq 0$. For our purposes the details of the nonlinear functions b_A and a_A are not important. Following the design of [42], we will assume that feedback linearization has already been carried so that:

$$\dddot{x}_A = u \tag{25}$$

The objective is to design a safe, comfortable and efficient controller for this linear system. It is assumed that safety takes precedence over the other two requirements. Comfort will be assumed to be more important than efficiency. Quantitative definitions of these design requirements will be given in Section 5.2.

[5] The controllers may actually depend on the set point. To avoid additional subscripts we will ignore this dependency in the notation.

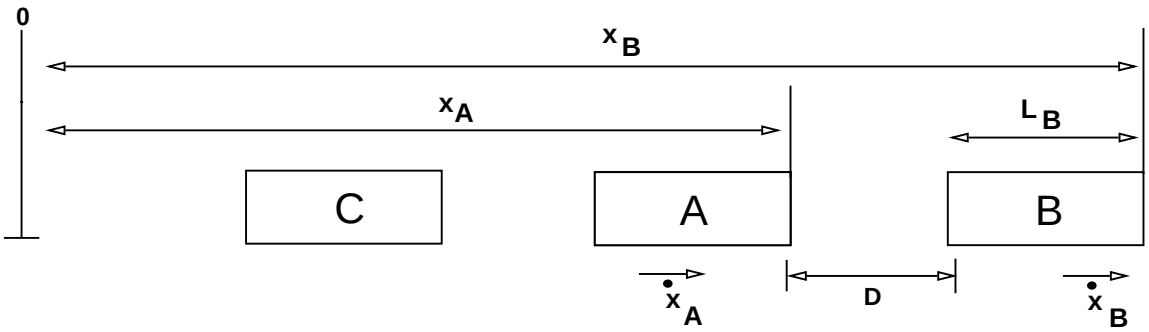

Figure 5: Vehicle Following

5.1 Hybrid Model Formulation

To remove the absolute position from the problem let $D = x_B - x_A - L_B$ denote the spacing between vehicles A and B. All pertinent information can now be encoded by the state vector $x = [\dot{x}_A \;\; \ddot{x}_A \;\; D \;\; \dot{D}]^T = [x_1 \;\; x_2 \;\; x_3 \;\; x_4]^T$ which evolves according to:

$$\begin{aligned} \dot{x} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \ddot{x}_B = Ax + Bu + D\ddot{x}_B \qquad (26) \\ x(0) &= x^0 \end{aligned}$$

For the vehicle following problem we are interested in regulating the spacing and relative velocity to a desired fixed point. In addition, whenever possible, a vehicle may be required to track a certain velocity, v_H, calculated by the roadside controllers in order to maximize throughput. These requirements can be encoded by means of three outputs:

$$y = \begin{bmatrix} D \\ \dot{D} \\ \dot{x}_A \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} x = Cx \qquad (27)$$

We assume that vehicle A has access to full state information for feedback control (a reasonable assumption under current sensor technology).

To complete the picture we also need to encode the state and input constraints imposed by the engine, tire and road conditions:

$$\begin{aligned} x(t) &\in \mathbf{X_C} = \left\{ x \in \mathbb{R}^4 | x_1 \in [v^A_{min}, v^A_{max}], x_2 \in [a^A_{min}, a^A_{max}], x_4 + x_1 \in [v^B_{min}, v^B_{max}] \right\} \\ u(t) &\in \mathbf{U} = [j_{min}, j_{max}] \\ \ddot{x}_B(t) &\in \mathbf{D} = [a^B_{min}, a^B_{max}] \end{aligned}$$

It is assumed that vehicles will not be allowed to go backwards, therefore $v^A_{min} = v^B_{min} = 0$ will be used. v^A_{max} is imposed by engine limitations. One of the objectives of the controllers we design will be fuel efficiency. As a consequence the engine will not have to be pushed to its limits for maximum speed and therefore v^A_{max} will not feature in the safety calculations. We

will assume $v^A_{max} = \infty$ to simplify the analysis. Typical values of a^A_{min}, etc. can be found in [21].

There are four kinds of exogenous inputs that influence the system evolution:

1. the jerk input of vehicle A, $u(t)$,
2. the trajectory of $\ddot{x}_B(t)$,
3. collisions of vehicle B with its preceding vehicle (that affect $\dot{x}_B(t)$ and hence x_4),
4. collisions of vehicle A with vehicle C (that affect x_1 and hence x_4).

$u(t)$ is assumed to be under the control of the designer. The remaining inputs will be treated as disturbances. The acceleration disturbance $\ddot{x}_B(t)$ can be modeled as a piecewise continuous function of time:

$$\ddot{x}_B : [0, \infty) \longrightarrow [d_{min}, d_{max}]$$

In hybrid automaton terminology this is a *continuous disturbance.*

The collision disturbances can be modeled by an instantaneous jump in the state of the system. We will assume that the relative velocity of all collisions is bounded in an interval $[v_a, 0]$ and all collisions are elastic. Then, a collision of vehicle B with the vehicle in front of it at time T_B with relative velocity δv_B will result in a reset of the state:

$$x_4(T'_B) := x_4(T_B) + \delta v_B$$

Similarly a collision between vehicles A and C at time T_C and with relative velocity δv_C will result in a reset of the state[6]:

$$\begin{aligned} x_1(T'_C) &:= x_1(T_C) - \delta v_C \\ x_4(T'_C) &:= x_4(T_C) + \delta v_C \end{aligned}$$

In the ensuing discussion we will be interested in situations where vehicle B and vehicle C can experience at most one collision. The requirement that vehicle B does not go backward after the collision can be encoded by restricting the set of allowable disturbances:

$$\begin{aligned} \mathcal{D} = \{d \quad | \quad & \ddot{x}_B(t) \in [d_{min}, d_{max}], \\ & 0 \le T_B, \delta v_B \in [\max\{v_a, x_4(T_B) + x_1(T_B)\}, 0] \\ & 0 \le T_C, \delta v_C \in [v_a, 0]\} \end{aligned} \tag{28}$$

Collisions between vehicles C and A can not lead to a violation of the state constraints, since we assume $v^A_{max} = \infty$. In hybrid system terminology the collisions are *discrete disturbances.*

Under these assumptions, the dynamical evolution of vehicle A is determined by a hybrid dynamical system. It is shown pictorially in Figure 6. The discrete state can take one of five values, *00, 01, 10, 11* and *CRASH.* The first four keep track of whether vehicles B and C have experienced a collision (1) or not (0). The discrete state *CRASH* is introduced to model the situation where vehicle A crashes into vehicle B. There are six continuous states, two keeping track of when the collision occurred and four keeping track of the state of vehicle A. The initial conditions, node invariants, transition guards, events and reset maps should be

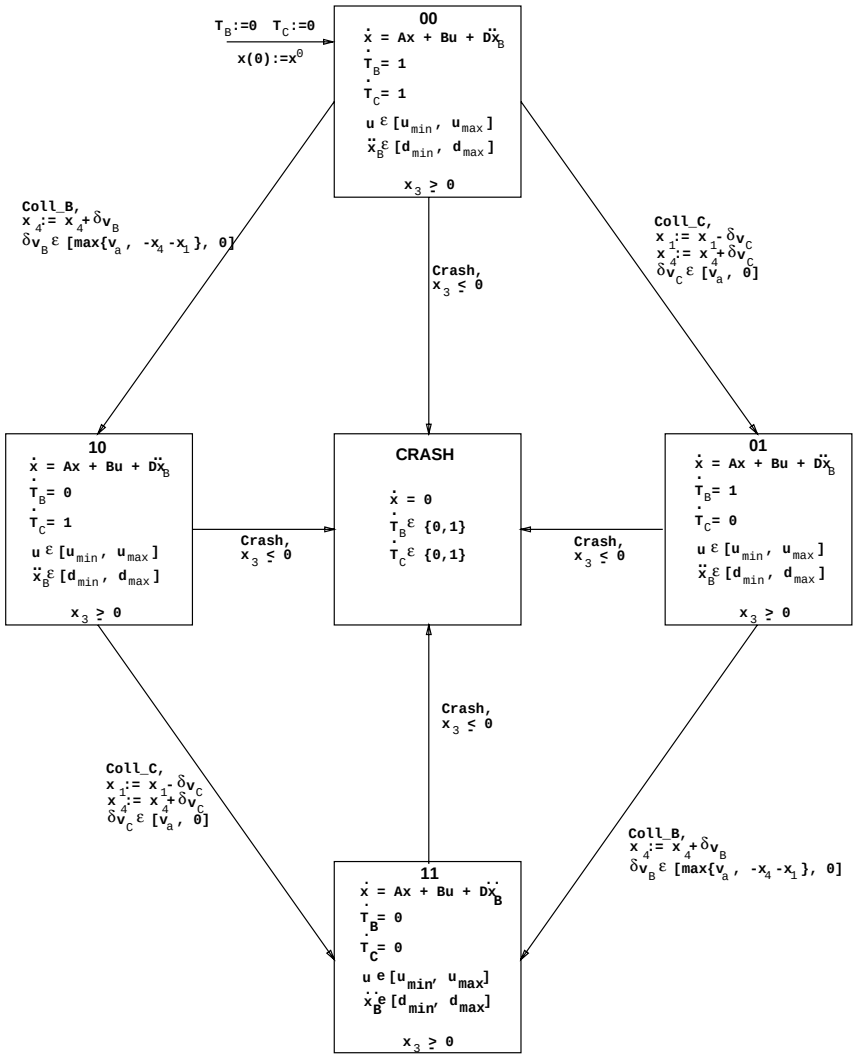

Figure 6: Hybrid Automaton for Automated Vehicle Operation

obvious from the figure. The automaton runs can be obtained analytically using the fact that A is nilpotent ($A^3 = 0$). Define the step function:

$$1_T(t) = \begin{cases} 0 & \text{if } t < T \\ 1 & \text{if } t \geq T \end{cases}$$

Then:

$$x(t) = \begin{bmatrix} x_1^0 + tx_2^0 \\ x_2^0 \\ -t^2 x_2^0/2 + x_3^0 + tx_4^0 \\ -tx_2^0 + x_4^0 \end{bmatrix} + \int_0^t \begin{bmatrix} t - \tau \\ 1 \\ -(t-\tau)^2/2 \\ -t + \tau \end{bmatrix} u(\tau) d\tau$$

[6]Note that in the coordinate system defined above $\delta v_B \leq 0$ and $\delta v_C \leq 0$.

$$+ \int_0^t \begin{bmatrix} 0 \\ 0 \\ t-\tau \\ 1 \end{bmatrix} \ddot{x}_B(\tau)d\tau + \begin{bmatrix} 0 \\ 0 \\ t-T_B \\ 1 \end{bmatrix} 1_{T_B}(t)\delta v_B + \begin{bmatrix} -1 \\ 0 \\ t-T_C \\ 1 \end{bmatrix} 1_{T_C}(t)\delta v_C$$

5.2 Design Requirements

The design problem can be viewed as a two player zero sum game, one player being the action of vehicle A and the other the disturbances (action of vehicle B and possible collisions). The two players compete over a number of cost functions, J_i, encoding various desirable properties of the system. All cost functions will be independent of the discrete state of the system ($q \in \{00, 01, 10, 11, CRASH\}$), therefore only the continuous state, x, will appear in the expressions.

1. **Safety (No Collision):**

$$J_1(x^0, u, d) = -\inf_{t \geq 0} x_3(t) \tag{29}$$

   A safe maneuver is one where:

$$J_1(x^0, u, d) \leq C_1 = 0 \; m$$

   Allowing $J_1 = 0$ meters makes the limiting case (where the vehicles just touch with zero relative velocity) acceptable.

2. **Comfort (Bounded Jerk):**

$$J_2(x^0, u, d) = \sup_{t \geq 0} |u(t)| \tag{30}$$

   A comfortable maneuver is one where:

$$J_2(x^0, u, d) \leq C_2$$

   The value $C_2 = 2.5 ms^{-3}$ is suggested in the transportation studies literature.

3. **Efficiency (Fast Convergence):**

$$J_3(x^0, u, d) = \int_0^\infty (y(\tau) - y_d)^T P (y(\tau) - y_d) d\tau \tag{31}$$

   where y_d is the desired fixed point for a given maneuver and $P \geq 0$. For steady state operation the fixed point used in [42] is:

$$y_d^L = \begin{bmatrix} \lambda_v v_H + \lambda_p \\ 0 \\ v_H \end{bmatrix}$$

   for $\lambda_v = 1$ second, $\lambda_p = 10$ meters.

5.3 Design for safety

The design of a safe controller can be posed as a reachability question on the hybrid automaton of Figure 6. To answer the safety question we would like to know under what conditions the discrete state *CRASH* is reachable. We will try to answer the question by determining the saddle solution (u_1^*, d_1^*) for the two player zero sum game with cost function J_1. Because of the nature of the cost function it is impossible to tackle this problem with conventional game theoretic techniques (Hamilton-Jacobi-Isaacs equations, Maximum Principle, etc.). Instead, we will try to guess the saddle solution and then show that it satisfies the definition. Consider the candidate saddle strategy, (u_1^*, d_1^*), given by:

$$\begin{aligned} u_1^*(t) &= \begin{cases} j_{min} & \text{if } t \le T_1 \\ 0 & \text{if } t > T_1 \end{cases} && (32) \\ d_1^*(t) &= \{\ddot{x}_B^*, (T_B^*, \delta v_B^*), (T_C^*, \delta v_C^*)\} && (33) \end{aligned}$$

where:

$$\begin{aligned} \ddot{x}_B^*(t) &= \begin{cases} d_{min} & \text{if } t \le T_2 \\ 0 & \text{if } t > T_2 \end{cases} \\ T_B^* &= 0 \\ \delta v_B^* &= \max\{v_a, x_4(T_B) + x_1(T_B)\} \\ T_C^* &= 0 \\ \delta v_C^* &= v_a \end{aligned} \qquad (34)$$

T_1 is the time when the acceleration of vehicle A reaches a_{min}^A under j_{min} and T_2 the time when vehicle B stops under d_{min}. Let T_3 be the time when vehicle A stops. Then:

$$T_1 = \frac{a_{min}^A - x_2^0}{j_{min}} \qquad (35)$$

$$T_2 = -\frac{\delta v_B^* + x_1^0 + x_4^0}{d_{min}} \qquad (36)$$

$$T_3 = \begin{cases} \frac{-x_2^0 - \sqrt{(x_2^0)^2 - 2j_{min}(x_1^0 - \delta v_C^*)}}{j_{min}} & \text{if } 0 \le T_3 \le T_1 \\ \frac{(a_{min}^A - x_2^0)^2 - 2j_{min}(x_1^0 - \delta v_C^*)}{2j_{min}a_{min}^A} & \text{if } T_1 \le T_3 \end{cases} \qquad (37)$$

Lemma 1 *(u_1^*, d_1^*) is globally a saddle solution for cost $J_1(x^0, u, d)$.*

Proof: For (u_1^*, d_1^*) to be a saddle solution we need to show that a unilateral change in strategy leaves the player who decided to change worse off. Let $x^*(t)$ denote the state at time t under the inputs (u_1^*, d_1^*). In particular:

$$\begin{aligned} x_3^*(t) &= \begin{bmatrix} 0 & -t^2/2 & 1 & t \end{bmatrix} x^0 - \int_0^t \frac{(t-\tau)^2}{2} u_1^*(\tau)d\tau + \int_0^t (t-\tau)\ddot{x}_B^*(\tau)d\tau \\ &\quad +(\delta v_B^* + \delta v_C^*)t \\ x_4^*(t) &= \begin{bmatrix} 0 & -t & 0 & 1 \end{bmatrix} x^0 - \int_0^t (t-\tau)u_1^*(\tau)d\tau + \int_0^t \ddot{x}_B^*(\tau)d\tau + (\delta v_B^* + \delta v_C^*) \end{aligned}$$

First, fix $d = d_1^*$ and let u vary. Let $x(t)$ be the state at time t under the inputs (u, d_1^*). Then:

$$x_4(t) - x_4^*(t) = \int_0^t (t-\tau)(u_1^*(\tau) - u(\tau))d\tau$$

We need to distinguish two cases. If $t \leq T_1$, the bounds on u imply that $u(\tau) \geq u_1^*(\tau)$, therefore $x_4(t) - x_4^*(t) \leq 0$. On the other hand, if $t \geq T_1$, recall that $\ddot{x}_B^*(t)$ is piecewise constant, with a discontinuity at T_2 (which may be either greater or less than T_1). Therefore $x_4(t)$ and $x_4^*(t)$ are piecewise differentiable, with derivatives $-x_2(t)$ and $-x_2^*(t)$ respectively. By definition of T_1, $x_2^*(t) = a_{min}^A \leq x_2(t)$ in the interval of interest. Therefore, as $x_4(0) = x_4^*(0)$, $x_4^*(t) \geq x_4(t)$. In either case $\dot{x}_3(t) = x_4(t) \leq x_4^*(t) = \dot{x}_3^*(t)$. Using the fact that $x_3^*(0) = x_3(0) = x_3^0$ and integrating we obtain $x_3(t) \leq x_3^*(t)$ for all t. Therefore:

$$J_1(x^0, u, d_1^*) \geq J_1(x^0, u_1^*, d_1^*) \tag{38}$$

Now fix u_1^* and let d vary. Let $x(t)$ be the state at time t under the inputs (u_1^*, d). Then:

$$x_4(t) - x_4^*(t) = \int_0^t (\ddot{x}_B(\tau) - \ddot{x}_B^*(\tau))d\tau + (\delta v_B 1_{T_B}(t) - \delta v_B^*) + (\delta v_C 1_{T_C} - \delta v_C^*)$$

Note that:

$$\left.\begin{array}{rcl} \delta v_C^* & = & v_a \\ \delta v_C & \geq & v_a \\ v_a & \leq & 0 \end{array}\right\} \Longrightarrow \delta v_C 1_{T_C}(t) - \delta v_C^* \geq 0$$

If $\delta v_B^* = v_a$, the same is true for the term $(\delta v_B 1_{T_B}(t) - \delta v_B^*)$. If $\delta v_B^* > v_a$, then $x_4^*(0) + x_1^*(0) = 0$ (recall that $T_B^* = 0$) and therefore $x_4^*(t) + x_1^*(t) \equiv 0$ (once vehicle B stops it never starts moving again under d_1^*). But, $x_4(t) + x_1(t) \geq 0$ (by the assumed state constraint) and $x_1(t) = x_1^*(t)$ (as $x_1(t)$ does not depend on δv_B). Therefore, if $\delta v_B^* > v_a$, $x_4(t) - x_4^*(t) \geq 0$ for all $t \geq 0$.

Overall, under the assumed constraints on δv_B and δv_C, either $x_4(t) \geq x_4^*(t)$ or the term outside the integral is positive. For the integral term we need to distinguish two cases. If $t \leq T_2$, the bounds on $\ddot{x}_B$ imply that $\ddot{x}_B(\tau) \geq \ddot{x}_B^*(\tau)$. If on the other hand, $t \geq T_2$, by definition of T_2, $\int_0^t \ddot{x}_B^*(\tau)d\tau = -(x_1^0 + x_4^0)$. The state constraints imply that $x_1(t) + x_4(t) \geq 0$ (vehicle B does not go backwards), therefore $\int_0^t \ddot{x}_B(\tau)d\tau \geq -(x_1^0 + x_4^0)$. Therefore, in either case $\int_0^t (\ddot{x}_B(\tau) - \ddot{x}_B^*(\tau))d\tau \geq 0$. The stopping time for B under d will always be greater than the stopping time under d_1^*. We are therefore able to conclude that $\dot{x}_3(t) = x_4(t) \geq x_4^*(t) = \dot{x}_3^*(t)$. Integrating this inequality from 0 to t and using the fact that $x_3^*(0) = x_3(0) = x_3^0$ leads to:

$$J_1(x^0, u_1^*, d) \leq J_1(x^0, u_1^*, d_1^*) \tag{39}$$

Combining inequalities (39) and (38):

$$J_1(x^0, u_1^*, d) \leq J_1^*(x^0) \leq J_1(x^0, u, d_1^*) \tag{40}$$

for all d and u. By definition, (u_1^*, d_1^*) is globally a saddle solution. $\square$

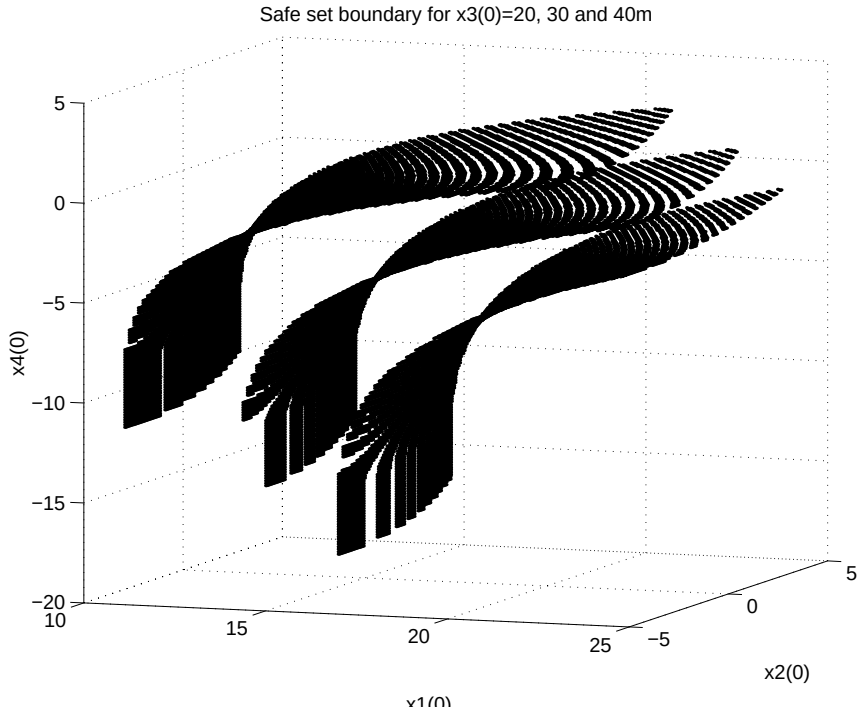

Figure 7: Safe set of initial conditions for $x_3^0 = 20, 30$ and 40 meters

5.3.1 Safe Set of Initial Conditions

Next we need to determine the set of initial conditions that are rendered safe by u_1^*. Recall that $x_3^*(t)$ is a continuously differentiable function of t defined on the compact interval $[0, T_3]$, with derivative $x_4^*(t)$. Therefore:

Lemma 2 *There exist* $\hat{T} \in [0, T_3]$ *such that* $J_1^*(x^0) = -x_3^*(\hat{T})$. *Moreover, either* $\hat{T} = 0$ *or* $\hat{T} = T_3$ *or* $x_4^*(\hat{T}) = 0$.

The calculations for analytically determining $J_1^*(x^0)$ from this lemma are rather messy. However, as the set of times T where $x_4^*(T) = 0$ is finite, we can easily carry out the calculation numerically. Figure 7 shows the set of points where $J_1^*(x^0) = 0$ for some values of x_3^0. Lemma 1 implies that the surfaces of Figure 7 are 2 dimensional slices of the 3 dimensional boundary of the safe set V_1 for various values of x_3^0. Any initial condition on or above these surfaces will not lead to a collision, provided $u = u_1^*$. The higher surfaces correspond to smaller initial spacings x_3^0 (the top to $20m$, the next to $30m$ etc.). As expected the safe set shrinks as x_3^0 decreases.

It should be noted that the safe set depends on the discrete state. Even though J_1 is independent of q, the set of allowable disturbances is larger at $q = 00$, smaller for $q \in \{01, 10\}$ and even smaller for $q = 11$[7]. In particular some collision disturbances are excluded in some discrete states. A simple corollary of Lemma 1 is:

[7] Clearly the safe set is empty if $q = CRASH$.

Corollary 1 *$(u_1^*, \{\ddot{x}_B^*, (t,0), (T_C^*, \delta v_C^*)\})$, $(u_1^*, \{\ddot{x}_B^*, (T_B^*, \delta v_B^*), (t,0)\})$ and $(u_1^*, \{\ddot{x}_B^*, (t,0), (t,0)\})$ are globally saddle solutions for J_1 if the discrete state becomes $q = 10, 01, 11$ at time t respectively.*

The effect of the reduced disturbance is to move the boundary of the safe set "down" (refer to Figure 7) by $|v_a|$ if $q \in \{01, 10\}$ and by $2|v_a|$ if $q = 11$.

We will not investigate these refinements further. We will define the **safe set** V_1 as the set of initial conditions for which the system is safe if $q = 00$, i.e.:

$$V_1 = \{x^0 \in \mathbf{X_C} | J_1^*(x^0) \leq C_1 = 0\} \tag{41}$$

V_1 still guarantees safety as it is contained in the safe sets for all q. It is not the best choice for a comfortable ride.

5.3.2 The Class of Safe Controls

The above analysis also allows us to determine the class of **safe controls** $\mathcal{U}_1(x^0)$, i.e. the controls for which any initial condition $x^0 \in V_1$ can not result in a crash. Let ∂V_1 denote the boundary of V_1 in the induced topology of $\mathbf{X_C}$ as a subset of $\mathbb{R}^4$. Define the **interior of** V_1 as:

$$\text{int}(V_1) = V_1 \setminus \partial V_1$$

Lemma 3 (Class of Safe Controls) *The class of safe controls for a given initial condition x^0 is given by:*

$$\begin{aligned} \mathcal{U}_1(x^0) &= \emptyset \text{ if } x^0 \in \mathbf{X_C} \setminus V_1 \\ u \in \mathcal{U}_1(x^0) &\iff \begin{cases} u(x) \in [j_{min}, j_{max}] & \text{if } x \in int(V_1) \\ u(x) = u_1^* & \text{if } x \in \mathbf{X_C} \setminus int(V_1) \end{cases} \end{aligned}$$

where u_1^* is given by equation (32) with x_2^0 being the acceleration of the vehicle at the time the boundary is reached.

Lemma 3 follows as a corollary of Lemma 1. The class of safe controls can be fully specified in feedback form (u_1^* is trivially a feedback controller). Notice that, if the discrete disturbances are removed, the control u_1^* is such that if the continuous state starts on the boundary of V_1 it will either "slide" along it or cross in the interior of V_1, depending on the acceleration of vehicle B. However, in the presence of discrete disturbances, V_1 is **not** an invariant set under u_1^*. A collision of vehicle B and/or C may push the trajectory well outside V_1. The effect of these collisions will be to change the discrete state to $q = 01$ or 10 or 11 (refer to Figure 6). Corollary 1 indicates that, after the collisions, the continuous state will still be in the safe set of the new discrete state. Overall, even though the continuous state trajectory crosses outside V_1 the system is still safe, as long as u_1^* is applied.

5.4 Design for comfort

Having established conditions for safety we can now improve the design by considering passenger comfort. We seek a saddle solution, (u_2^*, d_2^*) for $J_2(x^0, u, d)$. Consider:

$$u_2^*(x) = \begin{cases} u_1^* & \text{if } x \in \mathbf{X_C} \setminus \text{int}(V_1) \\ 0 & \text{otherwise} \end{cases} \tag{42}$$

As neither d nor x enter the cost function J_2, (u_2^*, d) will trivially be a saddle solution for every d. As before we can determine the set of initial conditions, V_2, for which the requirement for comfort can be satisfied. Assuming $C_2 < |j_{min}|$ this set is:

$$V_2 = \text{int}(V_1) \bigcup \{x \in \partial V_1 | x_2 = a_{min}^A\} \tag{43}$$

Moreover the class of comfortable controls for $x^0 \in V_2$ will be:

$$\mathcal{U}_2(x^0) = \left\{u \in \mathcal{U}_1(x^0) | u(t) \in [-C_2, C_2]\right\} \tag{44}$$

5.5 Design for efficiency

To complete the design the requirement for efficiency should also be addressed. We will not go into the details of the efficient design, as this problem can be approached in a number of ways and the solution does not affect safety in anyway. The saddle solution, (u_3^*, d_3^*), for cost function J_3 can be sought, for example. The result will be some form of H_∞ optimal design for u_3^*. Other designs (e.g. the one proposed in [42]) are also acceptable.

5.6 The Complete Controller

Consider the following switched feedback law:

$$u(x) = \begin{cases} u_3^* & \text{if } x \in V_2 \text{ and } u_3^* \in [-C_2, C_2] \\ C_2 & \text{if } x \in V_2 \text{ and } u_3^* > C_2 \\ -C_2 & \text{if } x \in V_2 \text{ and } u_3^* < -C_2 \\ u_1^* & \text{if } x \in \mathbf{X_C} \setminus V_2 \end{cases} \tag{45}$$

Note that the controller u can be easily encoded by a hybrid automaton. The structure can be directly inferred from equation (45) (Figure 8). This controller guarantees the safety of the system, provided the initial condition is chosen to be in the set V_1. In addition it guarantees that the vehicle operation will be comfortable and efficient, as long as safety is not compromised. The results of the design can be interpreted as saying that in the interconnection of the hybrid automata of Figures 6 and 8, the states corresponding to the state *CRASH* are unreachable.

6 Concluding Remarks

The control of large scale systems is one of the major problems facing control engineers today. In this paper, we introduced a rather general framework for design and analysis of hierarchical hybrid controllers for multi-agent large scale systems. The design algorithm presented in Section 4 provides a formal way of producing hybrid controllers that guarantee certain properties of the closed loop system by design, eliminating the need for verification. The conditions obtained by the application of our algorithm are, in some sense, necessary and sufficient. They are sufficient from the point of view of design. Any controller that satisfies the derived conditions is guaranteed to lead to acceptable performance for any disturbance. On the other hand, the conditions are necessary from the point of view of verification, in the

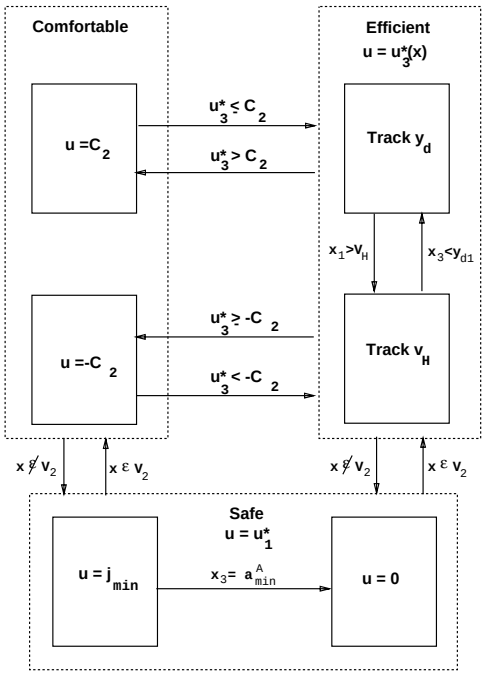

Figure 8: Controller Hybrid Automaton

sense of being "tight". Given a design that fails to satisfy these conditions, there exists a disturbance trajectory, d, and an initial condition, (q^0, x^0), allowed by the design, such that the trajectory generated by starting at (q^0, x^0) and applying the given controller and d violates at least one of the performance requirements.

An advantage of the approach presented here is that the calculations are flexible enough to be used in various contexts, other than controller design and verification. For example, they can be used to produce technological requirements on the physical plant (in particular the sensors and actuators) in order to guarantee certain levels of performance. This point of view is particularly evident in the automated highway problem [20]. There the solution to the game theory problems can be used to derive minimum sensor ranges, limits on the braking capability etc. needed to guarantee safe operation under certain requirements on throughput.

Besides technical conditions on the existence of saddle solutions for the games etc. the major problem that needs to be addressed in this context is the design of the discrete layer. The approach proposed here can be used to provide guidelines for the discrete design for semi-autonomous agent operation. The modifications to the initial discrete design, however, must be carried out by the designer possibly using standard discrete design and verification tools. The problem of parsing requirements on the emergent behavior (that are usually given linguistically) into sequences of way points, cost functions and thresholds also needs to be addressed. This problem is very difficult to formalize. The solutions given in applications are usually problem specific, require a tremendous amount of effort by the designer and are by no means formal. All these design issues are very interesting and deserve to be the topic of further research.

References

[1] R. F. Stengel, "Intelligent flight control systems," in *IMA Conference on Aerospace Vehicle Dynamics*, September 1992.

[2] A. Nerode and W. Kohn, "Multiple agent hybrid control architecture," in *Hybrid System* (R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, eds.), no. 736 in LNCS, pp. 297–316, New York: Springer Verlag, 1993.

[3] P. Varaiya and S. E. Shladover, "Sketch of an IVHS systems architecture," Tech. Rep. UCB-ITS-PRR-91-3, Institute of Transportation Studies, University of California, Berkeley, 1991.

[4] S. Sastry, G. Meyer, C. Tomlin, J. Lygeros, D. Godbole, and G. Pappas, "Hybrid systems in air traffic control," in *IEEE Control and Decision Conference*, pp. 1478–1483, 1995.

[5] K. M. Passino and P. J. Antsaklis, "Modeling and analysis of artificially intelligent planning systems," in *An Introduction to Intelligent and Autonomous Control* (P. J. Antsaklis and K. M. Passino, eds.), pp. 191–214, Boston: Kluwer Academic Publishing, 1993.

[6] I. Wagner and A. Bruckstein, "Cooperative cleaners: a study in ant robotics," Tech. Rep. 9512, CIS Report, Technion, IIT, Haifa, June 1995.

[7] A. Nerode and W. Kohn, "Models for hybrid systems: Automata, topologies, controllability, observability," in *Hybrid System* (R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, eds.), no. 736 in LNCS, pp. 317–356, New York: Springer Verlag, 1993.

[8] R. W. Brockett, "Hybrid models for motion control systems," in *Perspectives in Control* (H. Trentelman and J. Willems, eds.), Birkhäuser, 1993.

[9] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "An approach to the description and analysis of hybrid systems," in *Hybrid System* (R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, eds.), no. 736 in LNCS, pp. 149–178, New York: Springer Verlag, 1993.

[10] P. J. Antsaklis, J. A. Stiver, and M. Lemmon, "Hybrid system modeling and autonomous control systems," in *Hybrid System* (R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, eds.), no. 736 in LNCS, pp. 366–392, New York: Springer Verlag, 1993.

[11] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. H. Ho, "Hybrid automaton: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid System* (R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, eds.), no. 736 in LNCS, pp. 209–229, New York: Springer Verlag, 1993.

[12] M. S. Branicky, V. S. Borkar, and S. K. Mitter, "A unified framework for hybrid control: Background, model and theory," Tech. Rep. LIDS-P-2239, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1994.

[13] M. S. Branicky, *Control of Hybrid Systems*. PhD thesis, Massacussets Institute of Technology, 1994.

[14] A. Deshpande, *Control of Hybrid Systems*. PhD thesis, Department of Electrical Engineering, University of California, Berkeley, California, 1994.

[15] A. Puri, *Theory of Hybrid Systems and Discrete Event Systems*. PhD thesis, Department of Electrical Engineering, University of California, Berkeley, California, 1995.

[16] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata," in *STOCS*, 1995.

[17] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety*. New York: Springer-Verlag, 1995.

[18] D. N. Godbole, *Hierarchical Hybrid Control of Automated Highway Systems*. PhD thesis, Department of Electrical Engineering, University of California, Berkeley, California, 1994.

[19] J. Lygeros, D. N. Godbole, and S. Sastry, "Optimal control approach to multiagent, hierarchical system verification," in *IFAC World Congress*, 1996.

[20] J. Lygeros, D. N. Godbole, and S. Sastry, "A verified hybrid controller for automated vehicles." (preprint, submitted to Special Issue on Hybrid Systems of the IEEE Transactions on Automatic Control), March 1996.

[21] J. Lygeros, *Hierarchical Hybrid Control of Large Scale Systems*. PhD thesis, Department of Electrical Engineering, University of California, Berkeley, California, 1996.

[22] C. Tomlin, G. Pappas, and S. Sastry, "Conflict resolution for air traffic management: a case study in multi-agent hybrid systems." (preprint,submitted to Special Issue on Hybrid Systems of the IEEE Transactions on Automatic Control), 1996.

[23] P. Varaiya, "Smart cars on smart roads: problems of control," *IEEE Transactions on Automatic Control*, vol. AC-38, no. 2, pp. 195–207, 1993.

[24] A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya, "Protocol design for an automated highway system," *Discrete Event Dynamic Systems*, vol. 2, no. 1, pp. 183–206, 1994.

[25] T. Başar and P. Bernhard, *H^∞-Optimal Control and Related Minimax Design Problems*. Birkhäuser, 1991.

[26] J. C. Doyle, K. Glover, P. P. Khargonekar, and B. A. Francis, "State-space solutions to standard H_2 and H_∞ control problems," *IEEE Transactions on Automatic Control*, vol. 34, no. 8, pp. 831–847, 1989.

[27] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[28] B. S. Y. Rao and P. Varaiya, "Roadside intelligence for flow control in an IVHS," *Transportation Research - C*, vol. 2, no. 1, pp. 49–72, 1994.

[29] A. Hitchcock, "Casualties in accidents occuring during split and merge maneuvers," tech. rep., PATH Technical Memo 93-9, Institute of Transportation Studies, University of California, Berkeley, 1993.

[30] A. Deshpande, D. Godbole, A. Gollu, L. Semenzato, R. Sengupta, D. Swaroop, and P. Varaiya, "Automated highway system tool interchange format." (preprint) California PATH Technical Report, Institute of Transportation Studies, University of California, Berkeley, 1996.

[31] L. Semenzato, A. R. Deshpande, A. Gollu, P. Varaiya, D. Godbole, and R. Sengupta, "SHIFT reference manual." (preprint) California PATH Technical Report, Institute of Transportation Studies, University of California, Berkeley, 1996.

[32] F. Balarin, K. Petty, and A. L. Sangiovanni-Vincentelli, "Formal verification of the PATHO real-time operating system," in *IEEE Control and Decision Conference*, pp. 2459–2465, 1994.

[33] A. Puri and P. Varaiya, "Driving safely in smart cars," in *American Control Conference*, pp. 3597–3599, 1995.

[34] R. Alur and T. A. Henzinger, *Computer-Aided Verification*. 1996. to appear.

[35] L. Berkovitz, *Optimal Control Theory*. Springer-Verlag, 1974.

[36] T. Başar and G. J. Olsder, *Dynamic Non-cooperative Game Theory*. Academic Press, second ed., 1995.

[37] J. Lewin, *Differential Games*. Springer-Verlag, 1994.

[38] T. Başar and A. Haurie, eds., *Advances in Dynamic Games and Applications*, vol. 1 of *Annals of the International Society of Dynamic Games*. Birkhäuser, 1994.

[39] A. L. Schwartz, *Theory and Implementation of Numerical Methods Based on Runge-Kutta Integration for Solving Optimal Control Problems*. PhD thesis, Department of Electrical Engineering, University of California, Berkeley, California, 1996.

[40] J. Lygeros, D. N. Godbole, and M. E. Broucke, "Towards a fault tolerant AHS design." SAE Paper # 951894, Presented at SAE Future Transportation Technology Conference, Costa Mesa, 1995.

[41] D. N. Godbole, J. Lygeros, E. Singh, A. Deshpande, and A. Lindsey, "Design and verification of coordination layer protocols for degraded modes of operation of AHS," in *IEEE Control and Decision Conference*, pp. 427–432, 1995.

[42] D. N. Godbole and J. Lygeros, "Longitudinal control of the lead car of a platoon," *IEEE Transactions on Vehicular Technology*, vol. 43, no. 4, pp. 1125–1135, 1994.