Safety Verification for Automated Platoon Maneuvers: A Case Study

Ekaterina Dolginova and Nancy Lynch

MIT Laboratory for Computer Science
Cambridge, MA 02139, USA
{katya, lynch}@theory.lcs.mit.edu

Abstract. A system consisting of two platoons of vehicles on a single track, plus controllers that operate the vehicles, plus communication channels, is modeled formally, using the hybrid input/output automaton model of Lynch, Segala, Vaandrager and Weinberg [7]. A key safety requirement of such a system is formulated, namely, that the two platoons never collide at a relative velocity greater than a given bound v_{allow}. Conditions on the controller of the second platoon are given, designed to ensure the safety requirement regardless of the behavior of the first platoon. The fact that these conditions suffice to ensure safety is proved. It is also proved that these conditions are "optimal", in that any controller that does not satisfy them can cause the safety requirement to be violated. The model includes handling of communication delays and uncertainty. The proofs use composition, invariants, levels of abstraction, together with methods of mathematical analysis.
This case study is derived from the California PATH intelligent highway project, in particular, from the treatment of the platoon join maneuver in [3].

1 Introduction

Increasing highway congestion has spurred recent interest in the design of intelligent highway systems, in which cars operate under partial or total computer control. An important new effort in this area is the California PATH project (see, for example, [9]), which has developed a design for automating the operation of cars in several lanes of selected California highways. In this design, cars become organized into *platoons* consisting of a leader car and several following cars; the followers do not operate independently, but follow the control instructions of the leader.

An important maneuver for the proposed PATH system is the *platoon join* maneuver, in which two or more adjacent platoons combine to form a single platoon. The design of such a maneuver is described and analyzed in [3]. This maneuver involves both discrete and continuous behavior: discrete behavior appears in the form of synchronization and agreement among the controllers about the join process, plus communication among the various system components,

whereas continuous behavior appears in the motion of the cars. The combination forms a hybrid system of considerable complexity.

A key issue for the platoon join maneuver is its safety, represented by the requirement that cars never collide at too great a relative speed. In [3], a proof of such a safety property is outlined, for the specific platoon join maneuver given in that paper. The key to the proof turns out to be that the given maneuver always ensures that either (a) the platoons are sufficiently far apart that the second platoon can slow down sufficiently before hitting the first platoon, or (b) the relative speeds of the two platoons are already close enough.

Although the outline [3] gives the key ideas, from our point of view, it is incomplete as a safety verification. It does not include a complete model of all system components – in particular, the discrete components are not modeled. It does not seem to cover all cases that could arise: for instance, only some types of communication delay are handled, and uncertainties in the values of some parameters are not considered. The analysis contains informal "jumps" in which certain types of behavior are claimed to be the "worst possible", and then only these cases are analyzed carefully; however, it is not made clear how one can be sure that the claimed worst cases are in fact the worst. Another problem is that the analysis is presented for just the single maneuver, and is intertwined with the proofs of other properties for that maneuver (successful join, optimality of join time). However, it seems that the analysis should be decomposable, for example, proving the safety requirement in a way that allows the proof to apply to other maneuvers besides just the platoon join.

In previous work [7], Lynch, Segala, Vaandrager and Weinberg have developed a formal model, the *hybrid input/output automaton model*, for hybrid systems, together with associated proof techniques. These techniques include methods based on automaton composition, on invariant assertions, on levels of abstraction, and on mathematical analysis for reasoning about continuous behavior. They have developed methods of incorporating standard methods of analysis into automaton-based proofs. So far, these methods have been used to model and verify a variety of simple real-time systems, including several very simple maneuvers arising in automated transportation systems ([11], [10], [6]).

In this case study, we apply the hybrid I/O automaton model and its associated proof methods to the task of describing and verifying safety for the PATH platoon join maneuver. This is a more complex example than those previously considered using hybrid I/O automata. We aim for an accurate, complete model of the system, plus proofs that cover all cases and accommodate all realistic variations, including delays and uncertainties. Our safety proofs should apply as generally as possible, for instance, to other maneuvers besides platoon join. Our model should also be usable for proving other properties, such as successful join and optimality. The system and its proofs should admit decomposition into separate parts, as far as possible, and should be easy to extend.

In the work we have completed so far, we have made certain simplifications. Namely, we consider the case of two platoons only (as in [3]), and we consider uncertainties in only some of the parameter values. Moreover, we pretend that

the controllers control the cars' acceleration rather than their jerk (derivative of the acceleration). We intend to remove these restrictions in later work, and are designing our models and proofs to make such extensions easy.

For this simplified setting, we have succeeded in modeling the complete system, which consists of two platoons of cars on a single track, plus controllers that operate the cars, plus communication channels. We have formulated the safety requirement, namely, that the two platoons never collide at a relative velocity greater than a given bound v_{allow}. We have given conditions on the controller of the second platoon, designed to ensure the safety requirement regardless of the behavior of the first platoon, and we have proved that these conditions suffice to ensure safety. Our proofs cover all cases, and are sufficiently general to apply to other maneuvers besides platoon join. The proofs use discrete systems techniques, such as composition, invariants, and levels of abstraction. Additionally, the methods of mathematical analysis developed for proving invariance of state-space sets in [2] are used for reasoning about the continuous parts of the system.

In addition to proving safety, we also give results showing that the given conditions on the controllers are "optimal", in the sense that any controller that does not satisfy them can cause the safety requirement to be violated. The optimality results are proved using the same techniques (in particular, invariants and composition) that are used for the safety proof. Again, the optimality results apply to other maneuvers besides platoon join.

An alternative approach to proving safety for the platoon join maneuver, based on game theory, is presented in [5], [4]. There has been a large amount of prior work on modelling and verification of hybrid systems, as represented, for example, in the six previous workshops on hybrid systems. Nearly all of this work differs from ours in using either control theory methods, or else algorithmic techniques (e.g., decision procedures based on finite-state analysis). Other formal models for hybrid systems appear in [8], [1]; these differ from ours primarily in placing less emphasis on issues of external behavior, composition and abstraction.

We consider the research contributions of this paper to be: (a) The model and proof of safety for the platoon join (and other maneuvers). (b) The optimality result and its proof. (c) A demonstration of the power of hybrid I/O automata and its associated proof methods for reasoning about interesting hybrid systems. (d) A demonstration of the use of abstraction levels as a means of handling complexity.

2 HIOA Model

The Hybrid I/O Automata model presented in [7] is capable of describing both continuous and discrete behavior. The model allows communication among components using both shared variables and shared actions. Several HIOA techniques make them particularly useful in modeling and reasoning about hybrid systems. These include composition, which allows to form complex automata from simple

building blocks; implementation relations, which make it easy to use levels of abstraction when modeling complex systems; invariant assertions, which describe the non changing properties of the system.

A *state* of a HIOA is defined to be a valuation of a set of variables. A *trajectory* w is a function that maps a left-closed interval I of the reals, with left endpoint equal to 0, to states; a trajectory represents the continuous evolution of the state over an interval of time. An HIOA A consists of:

- Three disjoint sets of *input, output and internal* variables. Input and output variables together are called *external* variables.
- Three disjoint sets of *input, output and internal* actions.
- A nonempty set of *start states*.
- A set of *discrete transition*, i.e. (state, action, state) triples.
- A set of trajectories over the variables of A.

We now define executions of HIOAs. A *hybrid execution fragment* of A is a finite or infinite alternating sequence of trajectories and actions, ending with a trajectory if it is finite. An execution fragment records all the discrete changes that occur in an evolution of a system, plus the continuous state changes that occur in between. Hybrid execution fragments are called *admissible* if they are infinite. A *hybrid execution* is an execution fragment in which the first state is a start state. A state of A is defined to be *reachable* if it is the last state of some finite hybrid execution of A. A *hybrid trace* of a hybrid execution records only the changes to the external variables. Hybrid traces of an HIOA A (hybrid trace that arise from all the finite and admissible hybrid executions of A) describe its visible behavior.

HIOA A *implements* HIOA B if every behavior of A is allowed by B. A is typically more deterministic than B in both the discrete and the continuous level. Formally, if A implements B, then 1) A and B are *comparable* HIOA, meaning that they have the same external actions and external variables; 2) all the hybrid traces of A are included in those of B. To prove the second part, we need to show that there exists a *simulation relation* from A to B. A simulation relation from A to B is a relation R from states of A to states of B satisfiying:

- If s_A is a start state of A, then there exists s_B, $s_A R s_B$, such that s_B is a start state of B.
- If a is an action of A, (s_A, a, s'_A) is a discrete transition of A, $s_A R s_B$, and both s_A and s_B are reachable, then B has a finite execution fragment starting with s_B, having the same trace as the given step, and ending with a state s'_B with $s'_A R s'_B$.
- If w_A is a trajectory of A from s_A to s'_A, $s_A R s_B$, and both s_A and s_B are reachable, then B has a finite execution fragment starting with s_B, having the same trace w, and ending with a state s'_B with $s'_A R s'_B$.

Another technique for reducing complexity is HIOA *composition*. HIOAs A and B can be composed if they have no output actions or output variables in common, and if no internal variable of either is a variable of the other. The

composed HIOA C's input variables/actions are the union of A and B's input variables/actions minus the union of A and B's output variables/actions; all the other components (output and internal variables/actions, start states, discrete actions, trajectories) are the unions of the corresponding components of A and B. The crucial result is that the composition operator respects the implementation relation: if A_1 implements A_2 then A_1 composed with B implements A_2 composed with B. Finally, *invariant assertions* state system properties that are true in any reachable state of the system.

3 System Model

We consider two platoons of vehicles, moving along a single track. While the behavior of the leading platoon is arbitrary, the second platoon's controller must make sure that no "bad" collisions occur. "Bad" collisions are collision at a high relative speed. This is called the *Safety* requirement for the second controller. This *Safety* requirement is general for all platoon maneuvers, and is independent of the particular algorithm used. We devise the most nondeterministic safe controller, so that later we can use this controller as a correctness check: a controller implementing any platoon maneuver must implement our safe controller in order to be correct. This should be very useful in formally proving correctness of complicated algorithms.

3.1 *Controlled-Platoons*

We compose our system of a piece modeling the real world (the physical platoons) and two pieces modeling the controllers of each platoon (which are described in the next subsection). Each piece is modeled by a hybrid automaton. The real world piece is called *Controlled-Platoons*, shown in Figure 1. It consists of two platoons, named 1 and 2, where platoon 1 precedes platoon 2 on a single track. Positions on the track are labeled with nonnegative reals, starting with 0 as a designated beginning point. We pretend for simplicity here that the platoons have size 0. In the full version of the paper this restriction is relaxed. Note that the velocities of the platoons are always nonnegative – the vehicles will never go backwards, and the platoons are not allowed to bypass each other.

Only single collisions are modeled here. A special *collided* variable keeps track of the first occurrence of a collision. Before a collision, the platoons obey their respective controllers by setting the given acceleration. After a collision occurs, the platoons are uncoupled from the controllers and their velocities are set arbitrarily.

We use the constants $v_{allow} \in \mathsf{R}^{\geq 0}$ to represent the largest allowable velocity when a collision occurs, and $a_{min} \in \mathsf{R}^{\geq 0}$ to represent the absolute value of the maximum emergency deceleration. The platoons' position, velocity, and acceleration data is modeled by x_i, $\dot{x}_i$, and $\ddot{x}_i$, respectively. The dots are used as a syntactic device only. The differential relationships between these variables is a consequence of the trajectory definitions; however, this differential relationship

Actions:

Internal: *collide*

Variables:

Input: $\ddot{x_i} \in \mathsf{R}$, $i \in \{1,2\}$, initially arbitrary

Output: $\dot{x_i} \in \mathsf{R}^{\geq 0}$, $i \in \{1,2\}$, initially arbitrary

$x_i \in \mathsf{R}^{\geq 0}$, $i \in \{1,2\}$; initially $x_2 = 0$ and x_1 is arbitrary

collided, Boolean, initially *false*

now, initially 0

Discrete Transitions:

collide

Pre: $x_1 = x_2$

$collided = false$

Effect: $\dot{x_i}$:= arbitrary value, $i \in \{1,2\}$

$collided = true$

Trajectories:

an I-trajectory w is included among the set of nontrivial trajectories exactly if

collided is unchanged in w

for all $t \in I$ the following hold:

if $collided = false$ in w then

$w(t).\dot{x_i} = w(0).\dot{x_i} + \int_0^t w(u).\ddot{x_i} du$, $i \in \{1,2\}$.

$w(t).now = w(0).now + t$

$w(t).x_2 \leq w(t).x_1$

$w(t).x_i = w(0).x_i + \int_0^t w(u).\dot{x_i} du$

if $w(t).x_1 = w(t).x_2$ and t is not the right endpoint of I then

$collided = true$.

Fig. 1. The *Controlled-Platoons* Hybrid I/O Automaton

is partly lost after a collision occurs. The acceleration data is received from the controllers which are defined below. This will be used in our statement of the correctness property, below — we only want to assert what happens the first time a collision occurs. The second conditions on the trajectories of *Controlled-Platoons* guarantees that the platoon only executes the controller's decisions until the first collision occurs.

3.2 Controllers

$Controller_1$ is described in Figure 2. It is an arbitrary hybrid automaton with the given interface, restricted only by physical limitations. Note that the controller does not have any actions. The last restriction on continuous trajectories, for example, guarantees that the controller does not make the platoon to have negative velocity. The internal velocity and position variables ($\dot{x}_{int1}$ and x_{int1}) are used to keep track of the platoon's own data. This data is obtained by integrating their acceleration settings. Since there are no delays or uncertainties, these variables should correspond exactly to the actual position and velocity of the platoon.

The $Controller_2$ hybrid automaton is the same as $Controller_1$, except that

Variables:

Input: $\dot{x_2} \in \mathsf{R}^{\geq 0}$

$x_2 \in \mathsf{R}^{\geq 0}$

Output: $\ddot{x_1}$

Internal: $\dot{x}_{int1} \in \mathsf{R}^{\geq 0}$, initially $\dot{x}_{int1} = \dot{x}_1$

$x_{int1} \in \mathsf{R}^{\geq 0}$, initially $x_{int1} = x_1$

Trajectories:

an I-trajectory w is included among the set of nontrivial trajectories exactly if

$\ddot{x_1}$ is an integrable function

for all $t \in I$, at $w(t)$

$\dot{x}_{int1} = w(0).\dot{x}_{int1} + \int_0^t w(u).\ddot{x}_1 du$

$x_{int1} = w(0).x_{int1} + \int_0^t w(u).\dot{x}_{int1} du$

$\ddot{x}_1 \geq -a_{min}$

Fig. 2. *Controller*$_1$ Hybrid I/O Automaton

it inputs x_1 and $\dot{x}_1$, and outputs $\ddot{x}_2$.

Compose *Controlled-Platoons*, *Controller*$_1$ and *Controller*$_2$ using hybrid I/O automata composition rules to obtain an automaton that models our platoon system with each platoon having its own controller.

3.3 Safety Condition

We place a safety condition on states of *Controlled-Platoons*. The safety condition guarantees that if the platoons ever collide, then the first time they do so, their relative velocity is no more than v_{allow}. We formulate this condition formally as an invariant assertion:

Safety : If $x_1 = x_2$ and $collided = false$, then $\dot{x_2} \leq \dot{x_1} + v_{allow}$.

We define a new automaton, *Safe-Platoons*, to serve as a correctness specification. *Safe-Platoons* is exactly the same as *Controlled-Platoons* except that all the states are restricted to satisfy the safety condition.

We are supposed to design *Controller*$_2$ so that when it is composed in this way with arbitrary *Controller*$_1$, the resulting system satisfies the safety condition. Then we can say that it implements the *Safe-Platoons* automaton, using a notion of implements based on preserving hybrid traces. Here, the hybrid trace includes the output variables, which are the positions, velocities and accelerations of both platoons plus the *collided* flag. That is enough to ensure that the *Safety* condition of the spec carries over to the implementation.

4 The Ideal Case

4.1 The Model

We start with a treatment of the safety property in the ideal setting. This allows us to prove some important properties of the simpler model first, and then extend

them to the more complicated models via simulation mappings. By ideal setting we mean that there are no delays and/or uncertainties in either the sensor's data or the controller's directives. In the next few sections we will make the model more realistic by relaxing these restrictions. Also, in this abstract, we make the simplifying assumption that the platoons have size 0. In the full paper we show how to relax this restriction easily.

We define and prove correctness of a specific $Controller_2$, called C_2, which implements our safety condition, in Figure 3. This controller is very nondeterministic.

Definition:

$safe\text{-}measure = \max(x_1 - x_{int2} - \frac{(\dot{x}_{int2})^2 - (\dot{x_1})^2 - (v_{allow})^2}{2a_{min}}, \dot{x_1} + v_{allow} - \dot{x}_{int2})$

Variables:

Input: $\dot{x_1} \in \mathsf{R}^{\geq 0}$
$x_1 \in \mathsf{R}^{\geq 0}$

Output: $\ddot{x_2}$, initially if $safe\text{-}measure \leq 0$, then $\ddot{x_2} = 0$

Internal: $\dot{x}_{int2} \in \mathsf{R}^{\geq 0}$, initially $\dot{x}_{int2} = \dot{x}_2$
$x_{int2} \in \mathsf{R}^{\geq 0}$, initially $x_{int2} = x_2$

Trajectories:

an I-trajectory w is included among the set of nontrivial trajectories exactly if
w is a trajectory of $Controller_2$
if $collided = false$ in $w(0)$ then $\forall t \in I$
if $safe\text{-}measure \leq 0$ then $\ddot{x_2} = -a_{min}$

Fig. 3. C_2 Hybrid I/O Automaton

C_2 ensures that if the position and velocity parameters are on the boundary defined by *safe-measure*, then platoon 2 is guaranteed to be decelerating as fast as possible. This is guaranteed by the second condition on the trajectories of C_2.

4.2 Correctness of C_2

We will now prove correctness of our controller. This means that any controller that implements C_2, will be correct (safe).

We define a predicate S on states of *Platoons*, as follows:

Predicate S: If $collided = false$ then $safe\text{-}measure \geq 0$, where

$$safe\text{-}measure = \max(x_1 - x_{int2} - \frac{(\dot{x}_{int2})^2 - (\dot{x_1})^2 - (v_{allow})^2}{2a_{min}}, \dot{x_1} + v_{allow} - \dot{x}_{int2})$$

This says (from the definition of *safe-measure*, see Figure 3) that if the platoons have not collided yet, then either (a) the distance between the two platoons is great enough to allow platoon 2 to slow down sufficiently before hitting platoon 1, even if platoon 1 decelerates at its fastest possible rate, or (b) the relative velocities of the two platoons are already close enough.

We define a new automaton *C-Platoons*, which is exactly like *Controlled-Platoons*, with the additional restriction that in all the initial states *safe-measure* ≥ 0 (thus all the initially states satisfy Predicate S, since initially *collided* = *false*). The system *Implemented-Platoons* that we are considering is the composition of *C-Platoons*, an arbitrary $Controller_1$, and C_2. C_2 is designed to guarantee explicitly that if S is ever violated, or even if it in danger of being violated (because equality holds), platoon 2 is decelerating as fast as possible. We claim that this strategy is sufficient to guarantee that S is always true:

Lemma 1. *S is true in every reachable state of the* Implemented-Platoons.

As a simple consequence of Lemma 1, we obtain the safety condition:

Lemma 2. *In any reachable state of Implemented-Platoons, if $x_1 = x_2$ and collided = false, then $\dot{x_2} \leq \dot{x_1} + v_{allow}$.*

Now we use Lemma 2 to prove that the system is in fact safe, i.e., that it implements *Safe-Platoons*. We prove this using a simulation relation f. This simulation is trivial – the identity on all state components of *Safe-Platoons* (velocities, positions, and the *collided* flag).

Lemma 3. *f is a forward simulation from the composed system* Implemented-Platoons *to Safe-Platoons.*

Proof: By induction on the number of steps in the hybrid execution. Lemma 2 deals with trajectories; the proofs for the start states and discrete steps are relatively simple.

Theorem 4. *The Implemented-Platoons system implements Safe-Platoons, in the sense that for every hybrid execution α of Implemented-Platoons, there is a hybrid execution α' of Safe-Platoons that has the same hybrid trace – here, means same positions, velocity and collided flag values.*

Proof. Implemented-Platoons and *Safe-Platoons* are comparable and by Lemma 3, there is a simulation relation f from *Implemented-Platoons* to *Safe-Platoons*. Therefore, this composed system implements *Safe-Platoons*.

4.3 Optimality

We will now prove optimality of *safe-measure* using the analysis theorem about non-increasing functions. Informally, we want to prove that any $Controller_2$ that does not implements C_2 is unsafe. The formal definition of this optimality property appears in Theorem 7. Combined with the correctness result of the previous subsection, this will allows to decide whether any given controller is safe, since it is safe *if and only if* it implements C_2.

Define $Controller_2$ to be bad (and call it $Bad\text{-}Controller_2$), if there exists some $Controller_1$, such that in any admissible hybrid execution α of an automaton composed of *Controlled-Platoons*, $Controller_2$ and $Controller_1$, $\exists s \in \alpha$, which does not satisfy Predicate S.

Define $Bad\text{-}Controller_1$, given $Bad\text{-}Controller_2$, so that in the system composed of $Bad\text{-}Controller_1$, $Bad\text{-}Controller_2$ and *Controlled-Platoons* (the system called *Bad-Platoons*), for any admissible hybrid execution β, the following hold:

- $\exists s \in \beta$, s does not satisfy Predicate S;
- strictly after the occurrence of s, $\ddot{x}_1 = -a_{min}$.

The first lemma shows that once Predicate S is violated, it will remain violated, given some "bad" $Controller_1$. Formally,

Lemma 5. *If a given $Controller_2$ is bad, then in any Bad-Platoons system with this $Controller_2$, Predicate S is violated in all the states $\in \beta$ that occur strictly after s, in which collided = false. (Bad-Platoons, β, s are as defined above.)*

The next lemma shows that if Predicate S is violated in some state, then *safety* will also be violated eventually. Formally,

Lemma 6. *If a given $Controller_2$ is bad, then in any Bad-Platoons system with this $Controller_2$, in any admissible execution γ, $\exists s' \in \gamma$, which does not satisfy* safety*).*

Theorem 7. *For any* Bad-Controller$_2$*, there always exists such* Controller$_1$ *(C_1', which is not necessarily the same as C_1), that a Bad-Platoons system composed with these controllers has in its hybrid trace a state s', in which safety is violated.*

The last theorem shows that our controller is optimal, i.e., any $Controller_2$ that does not implement it, might lead to an unsafe state, given some "bad" $Controller_1$.

5 Delayed Response

Now we consider the case where there is a delay between the receipt of information by the controller for platoon 2 and its resulting action. There appear to be two distinct types of delay to consider — the inbound and the outbound delay; we model them separately. The inbound delay is due to delays in communicating sensor information to the controllers. The outbound delay comes from the fact the controller's decision are implemented by the platoons after some delay.

We use levels of abstractions to deal with the complexity of the delayed case. The use of simulation relations enables us to build correctness and optimality proofs based on the previous ideal case results. This makes all the proofs significantly easier.

5.1 The System with Inbound and Outbound Delays

We model both the inbound and the outbound delays by special delay buffers. To obtain the delayed system, we then compose our new controller with the delay buffers. First, we introduce the inbound delay buffer B_i (lag time in communicating sensor information) in Figure 4.

Variables:

Input: $\dot{x_1} \in \mathsf{R}^{\geq 0}$, $x_1 \in \mathsf{R}^{\geq 0}$

Output: $\dot{x}_{i1} \in \mathsf{R}^{\geq 0}$, $x_{i1} \in \mathsf{R}^{\geq 0}$

Internal: $saved$ - maps from an interval $(0, d_i)$ to $(\dot{x_1}, x_1)$

Trajectories:

an I-trajectory w is included among the set of nontrivial trajectories exactly if for all $t \in I$, $t > 0$ the following hold:

$$w(t).(\dot{x}_{i1}, x_{i1}) = \begin{cases} w(0).saved(t) & \text{if } t < d_i \\ (w(t-d_i).\dot{x_1}, w(t-d_i).x_1) & \text{otherwise} \end{cases}$$

$\forall t' \in (0, d_i)$,

$$w(t).saved(t') = \begin{cases} w(0).saved(t'-t) & \text{if } t' > t \\ (w(t-t').\dot{x_1}, w(t-t').x_1) & \text{otherwise} \end{cases}$$

Fig. 4. B_i Hybrid I/O Automaton

B_i acts in such a way that the output variables have exactly the values of the input variables, exactly time d_i earlier, where d_i is the maximum "information delay" – the longest time that it can take for a controller to receive the velocity and position sensor data. This delay buffer actually implements the more realistic version, in which the length of the delay varies nondeterministically within known bounds. Initially, the buffer ($saved$) is "prefed" with information that could have happened in that initial time period (so that the last position and velocity values in the buffer match up the initial position and velocity values of the platoons). Setting the maximum deceleration for that "imaginary" time period lets the controller be the most flexible (and thus optimal, as will be proven later), in the initial d_i time period.

Formally, the initial value of the $saved$ variable is determined as follows. For any start state s of the system, construct a trajectory w of length d_i of *Controlled-Platoons* so that $w(0).\dot{x}_1 = s.\dot{x}_1 + d_i a_{min}$, $w(0).x_1 = s.x_1 + s.\dot{x}_1 d_i + \frac{a_{min} {d_i}^2}{2}$, and $\forall t \in (0, d_i)$, $w(t).\ddot{x}_1 = -a_{min}$. The second platoon's state components can be arbitrary, as long as no collisions occur. Now, $\forall t \in (0, d_i)$, let $saved(t) = w(t).(\dot{x}_1, x_1)$.

Next, we add an outbound delay buffer B_o (lag time in communicating control information). An outbound delay buffer B_o, is almost like the inbound buffer B_i, but with input variable $\ddot{x}_{o2}$, and output variable $\ddot{x}_2$. The $saved$ variable is the same as in B_i, except that it now "saves" $\ddot{x}_{o2}$ and the length of the interval is d_o, where $d_o \in \mathsf{R}^{\geq 0}$ is the maximum "action delay" – the longest time that it can take

for a platoon to react to the controllers directives. Again, the delay time-length is exact. Initially, $\forall t \in (0, d_o)$, $saved(t) = -a_{min}$. This makes the platoons safe in the initial d_o time interval even if the first platoon starts decelerating.

Definition:

$$safe\text{-}measure_d = \max(x_{i1} + \dot{x}_{i1}t' - \frac{a_{min}t'^2}{2} - x_{new2} - \frac{\dot{x}_{new2}^2 - (\dot{x}_{i1} - a_{min}t_1)^2 - (v_{allow})^2}{2a_{min}},$$
$$\dot{x}_{i1} - a_{min}t' - \dot{x}_{new2} + v_{allow}),$$

where $t' = \min(d_i + d_o, \frac{\dot{x}_{i1}}{a_{min}})$

Variables:

Input: $\dot{x}_{i1} \in \mathsf{R}^{\geq 0}$
$x_{i1} \in \mathsf{R}^{\geq 0}$

Output: $\ddot{x}_{o2}$, initially if $safe\text{-}measure \leq 0$, then $\ddot{x_2} = 0$

Internal: internal variables of $Controller_2$ ($\dot{x}_{int2}$ and x_{int2})
a_2 - maps from an interval $(0, d_o)$ to $\ddot{x}_{o2}$,
initially, $\forall t \in (0, d_o), a_2(t) = -a_{min}$, otherwise – arbitrary
x_{new2}, $\dot{x}_{new2}$ - the position and velocity of the second platoon
after time d_o passes, provided *collided* still equals *false*

Trajectories:

an I-trajectory w is included among the set of nontrivial trajectories exactly if
w is a trajectory of $Controller_2$
if $collided = false$ in $w(0)$ then for all $t \in I$, $t > 0$:
if $safe\text{-}measure_d \leq 0$ then
$\ddot{x_2} = -a_{min}$
$\forall t' \in (0, d_o)$,

$$w(t).a_2(t') = \begin{cases} w(0).a_2(t'-t) & \text{if } t' > t \\ w(t-t').\ddot{x}_{o2} & \text{otherwise} \end{cases}$$

$w(t).\dot{x}_{new2} = w(t).\dot{x}_{int2} + \int_0^{d_o} w(t).a_2(u)du$
$w(t).x_{new2} = w(t).x_{int2} + \int_0^{d_o} (\int_0^u w(t).a_2(u')du' + w(t).\dot{x}_{int2})du$

Fig. 5. D_2 Hybrid I/O Automaton

Finally, we modify the controller so that it handles the delays correctly. The controller D_2 (see Figure 5) implements $Controller_2$. It is similar to C_2 in that it also tries to keep the second platoon within the bounds set by $safe\text{-}measure_d$, which is *safe-measure* redefined for the delayed case. The new controller gets its inputs from the inbound delay buffer B_i and its output variable goes into the outbound delay buffer B_o. Additional internal variables (x_{new2} and $\dot{x}_{new2}$) are added to store the "future" position and velocity data, as calculated from the acceleration settings. A buffer for storing acceleration settings that the controller has output, but that has not been executed yet (a_2) is used for this purpose. Also, $safe\text{-}measure_d$ is defined instead of *safe-measure*; the only changes from *safe-measure* are that the 1st platoon's parameters are exchanged by their "worst-case" values after $d_i + d_o$ time units pass; and the 2nd platoon's parameters are exchanged by their projected values after d_o time units pass.

5.2 Correctness of D_2

We will now compose B_i, D_2, B_o using the hybrid I/O automata composition rules to obtain the delayed controller, which we call *Buffered-Controller*. A straightforward simulation relation shows that this composed system implements C_2. This simulation relation f is the identity on all the external state components of C_2. The use of simulation relations will allows us to prove correctness of our more complicated delayed controller relatively easily, since we have already proven correctness in the simple (ideal) case.

First we prove that if the old *safe-measure* (the one used in the ideal case) is non-positive in some state of a trajectory of *Buffered-Controller*, then the new controller D_2 (the one that has both the inbound and the outbound delays), will also output maximum deceleration, just as the old (ideal) controller would. Formally,

Lemma 8. *If collided* $= false$ *in* $w(0)$ *of a trajectory* w *of Buffered-Controller, then* $\forall t \in I$*, such that* $w(t).$*safe-measure* ≤ 0*,* $\ddot{x_2} = -a_{min}$*.*

Lemma 9. *f (an identity relation on all the external components of C_2) is a forward simulation from the composed system Buffered-Controller to C_2.*

Proof. By induction on the number of steps in the hybrid execution. Start states and discrete steps are proven trivially; Lemma 8 is used to prove the simulation relation on continuous trajectories.

This lemma proves that *Buffered-Controller* implements C_2, since the two automata are comparable and there is a simulation relation from the first one to the second one. Therefore, we are now able to prove correctness of our delayed controller:

Theorem 10. *The doubly-delayed hybrid automaton composed of C-Platoons, Buffered-Controller implements Safe-Platoons.*

Proof. We have proved that the system *Buffered-Controller* implements C_2 in Lemma 9. But by Theorem 4, C_2 composed with *C-Platoons* (the *Implemented-Platoons* system) implements *Safe-Platoons*. Thus, the doubly-delayed hybrid automaton composed of *C-Platoons*, and *Buffered-Controller* also implements *Safe-Platoons* by the hybrid I/O automata composition rules!

5.3 Optimality

We will now prove optimality of D_2. Again, we will be basing our proofs on the optimality property of the controller C_2, which was proven in section 4.3. We want to prove that any controller with both inbound and outbound delays that does not implement controller D_2, is unsafe given some "bad" controller C_1. However, knowing that C_2 is optimal makes the proof much easier: we only need to show that a controller that would let *safe-measure*$_d$ get negative, will

eventually lead to a state in which *safe-measure* itself is negative. Then we can use optimality of C_2 to show that any such controller would not be correct.

Define *Buffered-Controller*$_2$ to be bad (*Bad-Buffered-Controller*$_2$), if there exists some *Controller*$_1$ (C_{d1}), such that in any admissible hybrid execution α of an automaton composed of *Controlled-Platoons*, C_{d1} and *Bad-Buffered-Controller*$_2$, $\exists s_d \in \alpha$, which does not satisfy Predicate S_d.

Lemma 11. *Any Bad-Buffered-Controller*$_2$ *implements Bad-Controller*$_2$*.*

Since we have just shown that the delayed automaton implements the non-delayed one, we can use the optimality property of the ideal case controller, to prove the optimality of the delayed controller easily:

Theorem 12. *Given any* Bad-Buffered-Controller$_2$*, we can always construct* Controller$_1$ *(*C_{d1}*) such that a system composed of* Controlled-Platoons *and these controllers has in any admissible hybrid execution a state* s'*, in which* safety *is violated.*

Proof. Take any *Bad-Buffered-Controller*$_2$. By Lemma 11, it implements *Bad-Controller*$_2$. Then by Theorem 7, there exists such *Controller*$_1$ (C_1'), that a system composed of *Controlled-Platoons*, C_1' and this *Bad-Buffered-Controller*$_2$ has in its hybrid trace a state s' that violates *safety*.

Therefore, the delayed *Controller*$_2$ is also optimal.

6 Uncertainty

Our model already includes both the inbound and the outbound delays in sending and receiving information between the controller and *Controlled-Platoons*. Now we will introduce an extra complexity which will make the model even more realistic: the uncertainty in information that the controller receives. This inbound uncertainty arises from inexact sensors that communicate the position and velocity data to the controllers. We will use similar methods to the ones used in the delay case. A special "uncertainty buffer" automaton will be defined, similar to the previous delay buffers. Then, the uncertainty will be implemented by adding this new automaton to the model and modifying the controller slightly. We will then prove correctness using the simulation relation to the delayed case which we have already worked out. This use of levels of abstraction makes the proofs for the complicated case, which involves both the delays and the uncertainties, relatively easy to both write and understand.

6.1 The System

We implement the delayed controller D_2 with a composition of two hybrid automata: another controller U_2, and an inbound uncertainty buffer U_i. We call

Variables:

Input: $\dot{x}_{i1}$, x_{i1}

Output: $\dot{x}_{iu1}$, x_{iu1}

Trajectories:

an I-trajectory w is included among the set of nontrivial trajectories exactly if for all $t \in I$, $t > 0$ the following hold:

$\dot{x}_{iu1} \in [\dot{x}_{i1} - \dot{\delta}, \dot{x}_{i1} + \dot{\delta}]$

$x_{iu1} \in [x_{i1} - \delta, x_{i1} + \delta]$

Fig. 6. U_i Hybrid I/O Automaton

this composed system *Uncertain-Controller*. The uncertainty buffer U_i nondeterministically garbles the position and velocity data within the given bounds (see Figure 6). The bounds are predefined constants $\delta \in \mathsf{R}^{\geq 0}$ — the maximum absolute value of uncertainty in position sensor data, and $\dot{\delta} \in \mathsf{R}^{\geq 0}$ — the maximum absolute value of uncertainty in velocity sensor data. The controller U_2 is the same as D_2 except that it now takes its inputs from the inbound uncertainty buffer U_i and that *safe-measure*$_u$ (see below) is defined to account for the uncertainties. Same as in the delay case, the only changes from *safe-measure*$_d$ are that the first platoon's data is adjusted to the "worst case" behavior of the first platoon.

$$\text{safe-measure}_u = \max((x_{iu1} - \delta) + (\dot{x}_{iu1} - \dot{\delta})t'' - \frac{a_{min}t''^2}{2} - x_{new2} - \frac{(\dot{x}_{new2})^2 - ((\dot{x}_{iu1} - \dot{\delta}) - a_{min}t'')^2 - (v_{allow})^2}{2a_{min}}, (\dot{x}_{iu1} - \dot{\delta}) - a_{min}t'' - \dot{x}_{new2} + v_{allow}),$$

where $t'' = \min(d_i + d_o, \frac{\dot{x}_{iu1} + \dot{\delta}}{a_{min}})$.

6.2 Correctness of U_2

A straightforward simulation relation shows that the *Uncertain-Controller* system implements D_2. This simulation relation f is the identity on all state components of D_2.

First we show that if the old *safe-measure*$_d$ (the one used in the delayed case) is non-positive in some state of a trajectory of *Uncertain-Controller*, then the new controller (the one that has an inbound uncertainty), will also output maximum deceleration, same as the old (delayed only) controller would. Formally,

Lemma 13. *Let w be an I-trajectory of* Uncertain-Controller. *If collided = false in $w(0)$, then $\forall t \in I$, such that safe-measure$_d \leq 0$, $\ddot{x}_{o2} = -a_{min}$ and safe-measure$_u \leq 0$.*

Then we are able to prove that f is the simulation relation from the composed system *Uncertain-Controller* to D_2.

Theorem 14. *f (an identity relation on all the external state components of D_2) is a forward simulation from the Uncertain-Controller system to D_2.*

We have already proven correctness of the delayed controller D_2, and we have also shown that *Uncertain-Controller* implements D_2; thus, our new uncertain system is also correct in a sense that it implements our correctness specification, *Safe-Platoons*.

7 Conclusion

The system consisting of two platoons moving on a single track has been modeled using hybrid I/O automata, including all the components (physical platoons, controllers, delay and uncertainty buffers), and the interactions between them. Safety conditions were formulated using invariant assertions. Correctness and optimality of controllers were proved using composition, simulation mappings and invariants, and the methods of mathematical analysis. Complexity (delays and uncertainty) was introduced gradually, using the levels of abstraction, which significantly simplified the proofs.

The case study describes formally a general controller that would guarantee the safety requirement regardless of the behavior of the leading platoon. Such a controller can be later reused to prove correctness of complicated maneuvers, such as merging and splitting, where the setup is similar.

In future work, we will extend the model to handle outbound uncertainty; use jerk instead of acceleration; motion in 2D planes. Also, we will consider cases with several platoons operating independently. Additional properties of the join maneuver, such as successful join, time optimality, and passenger comfort, will be studied; other maneuvers arising in this setting will be investigated using the same models.

References

1. R. Alur, C. Courcoubetis, T.A. Henzinger, P.H. Ho, X. Nicollin, A. Olivero, J Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
2. Michael S. Branicky, Ekaterina Dolginova, and Nancy Lynch. A toolbox for proving and maintaining hybrid specifications. Submitted for publication. To be presented at *HS'96: Hybrid Systems*, October 12-16, 1996, Cornell University, Ithacs, NY.
3. Jonathan Frankel, Luis Alvarez, Roberto Horowitz, and Perry Li. Robust platoon maneuvers for AVHS. Manuscript, Berkeley, November 10, 1994.
4. John Lygeros. *Hierarchical Hybrid Control of Large Scale Systems.* PhD thesis, University of California, Department of Electrical Engineering, Berkeley, California, 1996.
5. John Lygeros, Datta N. Godbole, and Shankar Sastry. A game theoretic approach to hybrid system design. Technical Report UCB/ERL-M95/77, Electronic Research Laboratory, University of California Berkeley, October 1995.

6. Nancy Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. In *Proceedings of the Third AMAST Workshop on Real-Time Systems*, pages 1–22, Salt Lake City, Utah, March 1996.
7. Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O automata. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III: Verification and Control* (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of *Lecture Notes in Computer Science*, pages 496–510. Springer-Verlag, 1996.
8. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *REX Workshop on Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 447–484, Mook, The Netherlands, June 1991. Springer-Verlag.
9. Pravin Varaiya. Smart cars on smart roads: Problems of control. *IEEE Transactions on Automatic Control*, AC-38(2):195–207, 1993.
10. H. B. Weinberg and Nancy Lynch. Correctness of vehicle control systems: A case study. In *17th IEEE Real-Time Systems Symposium*, pages 62–72, Washington, D. C., December 1996. Complete version in Technical Report MIT/LCS/TR-685, Laboratory for Computer Science, Massachusetts Institute of Technology, February 1996. Masters Thesis.
11. H. B. Weinberg, Nancy Lynch, and Norman Delisle. Verification of automated vehicle protection systems. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III: Verification and Control* (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of *Lecture Notes in Computer Science*, pages 101–113. Springer-Verlag, 1996.

This article was processed using the LaTeX macro package with LLNCS style