Real-Time Reachability for Verified Simplex Design

TAYLOR T. JOHNSON, University of Texas at Arlington
STANLEY BAK, Air Force Research Laboratory
MARCO CACCAMO and LUI SHA, University of Illinois at Urbana-Champaign

The Simplex architecture ensures the safe use of an unverifiable complex/smart controller by using it in conjunction with a verified safety controller and verified supervisory controller (switching logic). This architecture enables the safe use of smart, high-performance, untrusted, and complex control algorithms to enable autonomy without requiring the smart controllers to be formally verified or certified. Simplex incorporates a supervisory controller that will take over control from the unverified complex/smart controller if it misbehaves and use a safety controller. The supervisory controller should (1) guarantee that the system never enters an unsafe state (safety), but should also (2) use the complex/smart controller as much as possible (minimize conservatism). The problem of precisely and correctly defining the switching logic of the supervisory controller has previously been considered either using a control-theoretic optimization approach or through an offline hybrid-systems reachability computation. In this work, we show that a combined online/offline approach that uses aspects of the two earlier methods, along with a real-time reachability computation, also maintains safety, but with significantly less conservatism, allowing the complex controller to be used more frequently. We demonstrate the advantages of this unified approach on a saturated inverted pendulum system, in which the verifiable region of attraction is over twice as large compared to the earlier approach. Additionally, to validate the claims that the real-time reachability approach may be implemented on embedded platforms, we have ported and conducted embedded hardware studies using both ARM processors and Atmel AVR microcontrollers. This is the first ever demonstration of a hybrid-systems reachability computation in real time on actual embedded platforms, which required addressing significant technical challenges.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Embedded and Cyber-physical Systems

General Terms: Design, Verification

Additional Key Words and Phrases: Formal verification, hybrid systems, cyber-physical systems

ACM Reference Format:
Taylor T. Johnson, Stanley Bak, Marco Caccamo, and Lui Sha. 2016. Real-time reachability for verified Simplex design. ACM Trans. Embed. Comput. Syst. 15, 2, Article 26 (February 2016), 27 pages.
DOI: http://dx.doi.org/10.1145/2723871

1. INTRODUCTION

Modern cyber-physical systems are large complex systems of systems, for which arguments about the behavior of the whole system rely on guarantees about the individual

An earlier version of this work was presented at RTSS 2014 [Bak et al. 2014]. DISTRIBUTION A. Approved for public release; Distribution unlimited. (Approval AFRL PA case number 88ABW-2015-4618, 28 SEP 2015).

Authors' addresses: T. T. Johnson, 500 UTA Blvd., Box 19015, Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019-0015; email: taylor.johnson@acm.org; S. Bak, Air Force Research Laboratory AFRL/RQQA, 2210 Eighth St, Building 146, Room 300, Wright-Patterson AFB, OH 45433; M. Caccamo and L. Sha, Siebel Center for Computer Science, 201 N. Goodwin Avenue, Department of Computer Science, University of Illinois, Urbana, IL 61801; emails: {mcaccamo, lrs}@illinois.edu.

DOI: http://dx.doi.org/10.1145/2723871

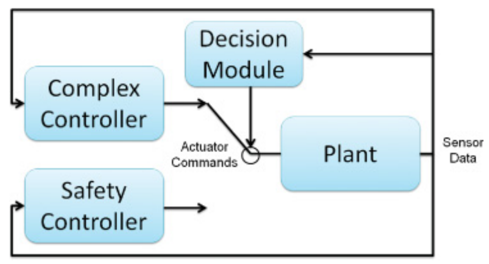

Fig. 1. The Simplex architecture produces a verified system despite the use of an unverified complex/smart controller. The decision module should switch between the controllers to provide overall system safety.

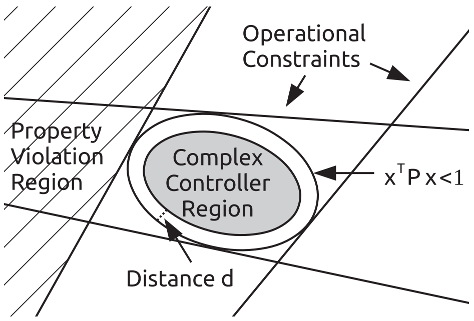

Fig. 2. The LMI Simplex design approach uses switching logic based on an ellipsoid within the system constraints in order to produce a verified system.

components. Individual components, however, may be designed using machine-learning methods such as neural networks, which are currently not amenable to formal analysis, or the components may simply be too large and complex for complete verification. As such autonomy is incorporated into these increasingly smart systems, which have the ability to learn from their environments and interactions through sophisticated complex/smart controllers, approaches are necessary to provide guarantees about their behavior.

One approach to provide formally verified behavior despite the use of unverified, complex, and smart control logic is the Simplex architecture [Sha 2001]. Similar to how a driving instructor's car may have two steering wheels and two sets of brakes, a Simplex system contains two controllers and supervisory switching logic. As long as the instructor intervenes to prevent dangerous situations, the untrusted student is allowed to drive. Similarly, in Simplex, an unverified controller can actuate the system as long as the verified one takes over quickly at potentially unsafe times.

In the Simplex architecture, shown in Figure 1, unverified control logic (the complex/smart controller) is wrapped with a verified controller (the safety controller) and switching logic (the decision module). The complex/smart controller typically has better performance, or is concerned with mission critical requirements, whereas the safety controller is designed with simplicity and provability in mind, and may concern itself only with safety-critical aspects. When the system is in danger of entering an unrecoverable state, the decision module must switch control to the safety controller. In this way, the complex/smart controller can be used while still maintaining the formal guarantees of the safety controller. The key challenge when designing a system with the Simplex architecture is to properly create the decision module logic.

It is easy to design safe decision-module switching logic; one can simply always use the safety controller. This is undesirable, however, as mission-critical objectives might be delayed or ignored since the complex/smart controller is never used. The key challenge, which is the focus of this article, is to *reduce the conservatism in the decision-module design*. Control should not be switched too late, though, as the safety controller may not be able to recover the system safely.

In earlier Simplex designs, the switching logic was designed in one of two ways. From a control theoretic perspective, verified switching logic can be synthesized from the solution of a linear matrix inequality (LMI) along with the system dynamics and constraints [Seto and Sha 1999]. Alternatively, approaches based on hybrid-systems reachability can be used to produce a provably safe decision module [Bak et al. 2011].

These earlier approaches are reviewed in Section 2. In this article, we propose the use of a unified approach, in which the offline LMI result is combined with an online reachability computation to produce a *significantly less conservative* Simplex system that is still safe. We elaborate on this approach and prove its safety in Section 3.

The proposed approach requires computing reachability online for short time intervals. Previous hybrid-systems reachability algorithms, however, were not designed for real-time computation and almost always require the use of numerous complex libraries for either performing simulations or for representing sets of reachable states as some geometric data structure (such as support functions, polytopes, zonotopes, symbolic expressions, and so on). For this reason, in Section 4, we propose a real-time reachability algorithm based on mixed face lifting [Dang 2000] that is compatible with the imprecise computation model in the real-time scheduling literature [Lin et al. 1987]. Real-time reachability has applications beyond Simplex, and is presented as a general online reachability approach. Next, we evaluate the proposed unified Simplex design in Section 5, on both x86 and embedded microprocessors. In order to provide a direct comparison, we use the existing system model from earlier Simplex work of an inverted pendulum system with saturation. The runtime approach significantly expands the space in which the complex/smart controller may be used. Other research efforts related to Simplex and reachability are presented in Section 6, followed by conclusions and directions for future work in Section 7.

2. BACKGROUND AND CONTRIBUTIONS

There have been several verified design methodologies for systems that use the Simplex architecture. Before going into their details, we first present useful definitions.

The system is defined with a set of operational constraints, such as limits of actuators, physical restrictions, invariant safety properties that cannot be violated, or linearization boundaries for which the model is considered valid.

Definition 1. States that do not violate any of the operational constraints are called *admissible states*. Those that violate the constraints are called *inadmissible states*.

From this definition, we can define the set of states that are recoverable for a particular control strategy, assumed to be a given safety controller in the Simplex architecture.

Definition 2. The set of *recoverable states* is a subset of the admissible states, such that, if the given safety controller is used from these states, all future states will remain admissible.

The recoverable states are used in the switching rule instead of the admissible states due effectively to inertia in the system. That is, they are used to ensure that the safety controller and actuators have enough time to prevent the system from leaving the admissible states. Further, the intuition of defining the recoverable states as a subset of the admissible states is as follows. To enhance performance, we wish to stay within a small subset of highly desirable admissible states. The set of recoverable states is the subset of the set of admissible states that a safety control is guaranteed not to leave. However, the safety controller may not be able to keep the system inside the subset of recoverable states, namely, the desirable states, hence the complex/smart controller is needed. Their relation is illustrated in Figure 2, in which the white ellipsoid is the recovery set.

With these definitions, we now describe two earlier approaches for verified Simplex design. The first is based on solving LMIs, and the second is based on reachability analysis of hybrid systems.

2.1. Verified Design Using LMIs

The first proposed way to design a verified decision module is based on solving LMIs [Seto and Sha 1999; Boyd et al. 1994], which has been used to design Simplex systems as complicated as automated landing maneuvers for an F-16 [Seto et al. 2000]. In this approach, system dynamics are approximated by a linear model using the standard control-theoretic approach, where $\dot{x} = Ax + Bu$ for state vector x and input u.

In this approach, the operational constraints and saturation limits are expressed as linear constraints in an LMI. These constraints, along with linear dynamics for the system, are input into a convex optimization problem that produces both linear proportional controller gains K as well as a positive-definite matrix P. The controller produced is a linear-state feedback controller, $u = Kx$, yielding the closed-loop dynamics $\dot{x} = (A+BK)x$. Given state x, when input Kx is used, the P matrix defines a Lyapunov potential function ($x^T Px$) that is positive-definite with a negative-definite derivative (thus, it is monotonically decreasing over time), guaranteeing stability of the linear system using Lyapunov's direct or indirect methods (if the plant is nonlinear and was linearized) [Khalil 2002]. Furthermore, the matrix P is constructed by the method such that it defines an ellipsoid in the state space in which all the constraints are satisfied when $x^T Px < 1$. Since the states in which saturation occurs were used as constraints in the method, any states inside the ellipsoid result in control commands that are not beyond the actuator limits (where saturation would occur).

In this way, when the gains K define the safety controller, the ellipsoid of states $x^T Px < 1$ is a subset of the recoverable states. The situation is shown in Figure 2. The feasible region is a subset of the admissible states defined by the input constraints (saturation), as well as the operational constraints. The stabilizable region (also known as the region of attraction) is the region of the state-space within which a given controller can stabilize the system. For the purpose of LMI-Simplex, this is also known as the recoverable region or the recoverable states, as defined in Definition 2. For linear systems with constraints, this region may be underapproximated by solving an LMI of the determinant maximization form [Vandenberghe et al. 1998]. For a matrix that describes an ellipsoid $x^T Px = 1$, this has the effect of maximizing the product of the radii of the ellipsoid (which is related to the determinant of the matrix P). The volume of an ellipsoid, then, is proportional to this product. In this way, the optimization is maximizing the volume of the ellipsoid such that all states inside do not leave the ellipsoid, and all the constraints are satisfied for every state in the ellipsoid.

This approach is used to determine the proper behavior of the decision module. As long as the system remains inside the ellipsoid, any unverified, complex/smart controller can be used. If the state approaches the boundary of the ellipsoid, control can be switched to the safety controller that will drive the system toward the equilibrium point where $x^T Px = 0$. Care must be taken to ensure that control is switched to the safety controller *before* the state leaves the ellipsoid. If the decision module simply checks the Lyapunov potential of the current state, then, once the state is outside of the ellipsoid, the system is not guaranteed to be recoverable without violating the operational constraints. Thus, a smaller subset of the state space must be used to define the states for which the complex controller is allowed to actuate the system. In Figure 2, the distance d defines this extra buffer that can be determined offline by computing the maximum gradient for any control command inside the ellipsoid, multiplied by the period of the decision logic. As long as d is no smaller than the maximum distance traveled in the state-space over the time of one full control period, then d is large enough to ensure that switching to the safety controller can recover the system.

For safety, it is sufficient to consider only a single switch to the safety controller and never switching back. If switching back is desired, this should not be done arbitrarily, as the composed switched system might be unstable. Specifically, the safety controller should be used at least until a state within the complex/smart controller region (as shown in Figure 2) is reentered, before switching back to the complex/smart controller.

2.2. Verified Design using Reachability

An alternative method for verified Simplex design is based on reachability analysis of hybrid systems [Bak 2013b], which has been used, for example, to create a Simplex system to prevent ofroad vehicle rollover [Bak et al. 2010]. In this approach, the dynamics are defined using a hybrid automaton, which is a formal model for a system with both continuous and discrete behaviors. Mathematically, a hybrid automaton [Alur et al. 1995] is a tuple, $H = (\mathcal{X}, L, X_0, I, F, T)$, where:

—$\mathcal{X}$ is the set of continuous states. For a system with n real-valued dimensions, the continuous state is $\mathbb{R}^n$.
—L is the set of discrete states (locations). The state of a hybrid automaton is an element of $X = L \times \mathcal{X}$.
—X_0 is a set of initial states, which is a subset of X.
—I is a set of invariants that defines the continuous states that are possible for each location. It is a function $L \to 2^{\mathcal{X}}$.
—F is a set of flows, each of which defines the differential equations in each location. It is a function $X \to 2^{\mathbb{R}^n}$.
—T is a set of discrete transitions, each of which defines switching between discrete locations. A transition is composed of a guard condition for when the transition is enabled, and a reset map that can reassign the continuous states from the predecessor mode to the successor mode. In general, it is a relation $T \subseteq X \times X$.

Semantically, a hybrid automaton behaves by advancing time according to the differential equations defined in the mode of the current discrete state $l \in L$, then allowing any enabled transitions to be taken, and repeating, yielding a sequence of states called an *execution*. A state $x \in X$ is *reachable* if there exists a finite execution ending in x. The *set of reachable states* contains every reachable state. The guard conditions on the outgoing transitions define when the location can change. The invariants of the locations can be used to force transitions by preventing time from elapsing further in the current mode. Together, these allow nondeterminism in the discrete behavior. A hybrid automaton can be graphically depicted as a finite-state machine with differential equations in each discrete state. The model also allows for nondeterminism in continuous behavior because a single state $x \in X$ may be associated with a set of derivatives for each variable, via the set of flows F.

This modeling framework is very expressive, and computing exactly the sets of states a hybrid automaton may enter, called the *reachable* set of states, is undecidable [Henzinger et al. 1995]. Thus, analysis of hybrid systems often restricts either the continuous dynamics or the discrete dynamics [Alur and Dill 1994; Lafferriere et al. 2000; Branicky 1998]. In this article, the reachability algorithm proposed in Section 4 considers restricted hybrid automata models in which (a) the state invariants are disjoint and cover the continuous states $\mathbb{R}^n$, (b) there are no reset maps in the transitions between discrete states, and (c) the guards of incoming transitions are defined by the state invariants.

In addition to restrictions on dynamics, practical reachability approaches often overapproximate the set of reachable states [Kapinski and Krogh 2002; Dang et al. 2010; Frehse et al. 2011], which is sufficient for proving safety properties. If a sound overapproximation of the reachable set of states for a hybrid automaton does not contain any unsafe states, then the system is verified as safe since no unsafe states are in the actual reachable set of states either. That is, the system is safe if the intersection of the overapproximation of the set of reachable states and the set of unsafe states is empty. However, this approach may lead to spurious counterexamples in which the error due to the overapproximation contains unsafe states, but the actual reachable set of states does not.

We define $\text{Reach}_{\infty}(x, \text{HA})$ to be the set of states reached *in any amount of time* from state x in hybrid automaton HA, $\text{Reach}_{\leq t}(x, \text{HA})$ is the set of states reached from x in *up to t* time, and $\text{Reach}_{=t}(x, \text{HA})$ is the set of states reached after *exactly t* time has elapsed. Also, we naturally extend Reach to initial *sets* of states, in which the resultant set of reachable states is the union of the set of reachable states from each state in the initial set.

In terms of Simplex design, the behavior of an optimal decision module can be defined in terms of reachability. Here, optimal means that the given safety controller takes over only if it has to; if it did not take over, then the system remains in admissible states and can enter the subset of recoverable states that can be precomputed offline, for example, using LMIs. Furthermore, it never takes over when the complex/smart controller could safely be used. The switching condition (formalized as the transition's guard and invariant in the hybrid automaton) between the safety controller and complex/smart controller modes is defined using the following theorem [Bak et al. 2011].

Theorem 1. *The optimal switching condition for Simplex is given when, at every control iteration, the complex/smart controller is used if and only if (1)* $\text{Reach}_{\leq\delta}(x, \text{CC}) \cap U = \emptyset$ *and (2)* $\text{Reach}_{\infty}(\text{Reach}_{=\delta}(x, \text{CC}), \text{SC}) \cap U = \emptyset$*, where $x \in X$ is the current state and $U \subseteq X$ is the set of inadmissible (unsafe) states.*

The inner $\text{Reach}_{=\delta}$ in Part (2) is the time-bounded reachability of the system for one decision logic switching interval time, δ, while using the complex/smart controller (CC). The outer Reach_{∞} is the infinite-time reachability for the system under control of the safety controller (SC).

Intuitively, this check is examining what happens if the complex/smart controller is used for a single control interval of time δ, then the safety controller is used thereafter. If this set of states contains an inadmissible state (either before the switch, as in Part (1), or after, as in Part (2)), then the complex/smart controller cannot be used for one more control interval; instead, the safety controller must be used right away. Assuming that the system starts in a recoverable state, this guarantees that it will remain in the recoverable set for all time.

Several factors prevent the direct use of Theorem 1. The first is that the reason to apply Simplex is that a precise model of the complex/smart controller is not available, but rather an overapproximation must be used that can be computed, for example, based on the plant model and actuator limits. Second, as discussed before, computing reachability exactly for a general hybrid automaton is undecidable. However, estimates of the set of recoverable states (Definition 2) can still be computed using overapproximations, where the conservativeness of the resultant decision module depends on the amount of overapproximation. Third, the switching condition is defined in terms of a specific state x, which is not useful for offline computation since every state would need to be enumerated. Instead, the condition can be rewritten in terms of backwards reachability from the set of inadmissible states, which can then be computed offline [Bak et al. 2011; Bak 2013b]. As with the LMI approach, the output is a set of states that forms a guaranteed subset of the recoverable states[1]. These considerations are

[1]The set of backward reachable states can be computed for deterministic systems (including linear systems) by negating the differential equations and inverting the transitions in the hybrid automaton, and using standard forward reachability techniques. This technique is known as back-reachability.

combined in order to provide a condition for effectively computing the decision-module logic as follows.

COROLLARY 2. *A safe switching condition for Simplex is given when, at every control iteration, the complex/smart controller is used if the current state* $x \notin \textsc{BackReach}^*_{\leq\delta}(\textsc{BackReach}^*(U, \textsc{sc}), \textsc{cc}')$.

Here, $\textsc{BackReach}^*$ is an overapproximation of the exact set of backward reachable states for all time (i.e., to a fixed point). The inner $\textsc{BackReach}^*$ defines the states in which, if the system were to start from the set of unsafe states U and use the safety controller, it could still violate one or more of the safety constraints. Then, the outer $\textsc{BackReach}^*_{\leq\delta}$ is the set of states that, within one control interval, can reach an unrecoverable state. Since the unrecoverable states contain the unsafe states, and since the outer $\textsc{BackReach}^*_{\leq\delta}$ checks up to δ time rather than exactly δ time, a separate condition is not needed to check if the complex/smart controller itself reaches the unsafe states, as in Part (1) of Theorem 1.

The pessimism in the resultant decision logic depends on both the accuracy of the reachability computation as well as on how much $\textsc{cc}'$ overapproximates the exact complex/smart controller model $\textsc{cc}$. The condition in Corollary 2 is more useful than the one in Theorem 1 because it can be effectively computed using existing hybrid-systems reachability algorithms. The set of states on the right-hand side can be computed offline and encoded in some form (e.g., using linear bounds [Bak 2009]) and then, online, the decision module need only check if the current state exists within the encoded set of states. If it does, then the safety controller must be immediately used. If it does not, then the complex/smart controller can be used for one control interval, after which the condition will be checked again on the new state.

2.3. Contributions

In this article, building on our prior work [Bak et al. 2014], we show how to combine the LMI-based Simplex method with a real-time reachability method into a unified framework to ensure safety while drastically decreasing the overconservative use of the safety controller. Specifically, if it is possible to use the set of recoverable states computed using the LMI method for the switching condition, we do so. If not and the system is at a state outside the recoverable states based on the LMI ellipsoids, then we try to check safety using a novel real-time reachability method, in contrast to the previous offline reachability approach. Together, we illustrate how this unified approach gives both real-time guarantees and reduces conservatism of when the safety controller is used. A main contribution of our approach is the first ever demonstration of a reachability method in real time, enabled by our careful design and implementation that does not use any dynamic memory allocation nor rely on sophisticated (nonportable) libraries that many other methods use, such as the Parma Polyhedral Library (PPL) [Bagnara et al. 2008], recent satisfiability-modulo theory (SMT) approaches [Gao et al. 2013], or validated integration tools [Duggirala et al. 2013]. To validate the feasibility of actually implementing the method in real-time embedded hardware, we have ported our prototype method from Bak et al. [2014] that was implemented on x86-64 platforms to several embedded platforms (i.e., a 32b, ARM-based system and an 8b, Atmel AVR ATmega32u4-based Arduino system). This effort validates our claims from Bak et al. [2014], which were not previously validated in embedded hardware. *The key result of this article is the first ever demonstration of a hybrid-systems reachability algorithm implemented in embedded hardware that can meet real-time guarantees, which required carefully designing the reachability algorithm as described in this article.* We have added significant further details of the approach and case study to the article over Bak et al. [2014], including code snippet examples for the case study.

3. UNIFIED APPROACH FOR SIMPLEX DESIGN

The two existing approaches for Simplex design previously discussed each have their own limitations. The LMI approach works when the system model is linear. If there are actuator limits, and the input to the actuators u (from $\dot{x} = Ax + Bu$) can saturate, the output of the optimization will be a set of states in which the command used by the safety controller is within the saturation limits. This is done by adding a constraint based on the state-feedback gain as part of the optimization (the input is $u = Kx$, which is bounded by the linear constraints $Kx \leq$ MAX_INPUT and $Kx \geq$ MIN_INPUT).

The set of states output by the LMI approach is safe but may be pessimistic, since a saturated safety controller may still be able to recover the system. Furthermore, the resultant switching condition is based on a Lyapunov function that—due to convexity and quadratic restrictions required in the optimization algorithms—has level sets that are ellipsoidal. This is a sufficient, but not necessary, condition for stability; therefore, the switching set is almost certainly conservative. We demonstrate this pessimism in our evaluation in Section 5.

The reachability-based Simplex approach is not restricted to linear systems, and can have its conservatism decreased by increasing the accuracy of the reachability computation[2]. One downside of this approach is that overapproximation error occurs from the need to abstract the complex/smart controller hybrid automaton by a hybrid automaton that takes into account any possible complex/smart controller command. A second issue is the difficulty of succinctly and accurately encoding the result of the computation, which, in general, may be a large nonconvex set in many dimensions. Last, hybrid-systems reachability methods introduce overapproximation error, which can be large when the initial set of states is large and the reachability time bound is large. The back-reachability formulation of Theorem 1 includes a time-unbounded reachability computation from the set of inadmissible states, which can require significant computation time.

We now present an alternative design for a verified Simplex system. The proposed technique makes use of aspects from both of the previous verified design approaches in order to overcome some of their individual limitations.

First, we formalize the connection of the ellipsoid from of the LMI approach with that of a reachability computation of a hybrid automaton (which, by the ellipsoid's construction, remains in a single, unsaturated mode):

LEMMA 3. *The output of the LMI approach, the potential function P, and controller gains K, define a safety controller* SC *and a subset of the recoverable set of states $\mathcal{R} = \{x | x^T Px < 1\}$, where* $\text{REACH}_\infty(\mathcal{R}, \text{SC}) \cap U = \emptyset$.

This is true because the potential function is guaranteed to satisfy the constraints passed to the LMI solver, including avoidance of the inadmissible states, when $X^T PX < 1$. When the controller gain vector K output by the approach is used (which defines the safety controller update $u = Kx$), the potential function is strictly decreasing over time (i.e., it is a Lyapunov function). Therefore, it is guaranteed for unbounded time that any state starting inside $\mathcal{R}$ will remain inside $\mathcal{R}$. Since there are no inadmissible states in $\mathcal{R}$, no inadmissible states will ever be reached.

We can now define an alternate condition for safe switching logic:

[2]Overapproximating reachability approaches typically results in an accuracy/computation time trade-off.

THEOREM 4. *A safe switching condition for Simplex is given when, at every control iteration, the complex/smart controller is used if, for some α time, (1)* $\textsc{Reach}_{\leq\delta}(x, \textsc{cc}) \cap U = \emptyset$, *(2)* $\textsc{Reach}_{\leq\alpha}(\textsc{Reach}_{=\delta}(x, \textsc{cc}), \textsc{sc}) \cap U = \emptyset$, *and (3)* $\textsc{Reach}_{=\alpha}(\textsc{Reach}_{=\delta}(x, \textsc{cc}), \textsc{sc}) \subseteq \mathcal{R}$.

PROOF. Intuitively, this switching condition states that the complex/smart controller may be used if: (1) the complex/smart controller cannot reach an unsafe state before the next decision interval (at time δ), (2) if the safety controller takes over at the next decision interval, it will avoid unsafe states until $\delta + \alpha$ time passes, and (3) after $\delta + \alpha$ time, a state in $\mathcal{R}$ will be reached.

More formally, assume by contradiction that this is not a safe switching condition; thus, an inadmissible state is reached at some time. This time will be either less than δ, more than δ, and less than $\delta + \alpha$, or more than $\delta + \alpha$. The first two of these cases are ruled out directly by Conditions (1) and (2), so only the third case needs to be examined.

From Lemma 3, $\textsc{Reach}_{\infty}(\mathcal{R}, \textsc{sc}) \cap U = \emptyset$. If $\mathcal{R}' \subseteq \mathcal{R}$, $\textsc{Reach}_{\infty}(\mathcal{R}', \textsc{sc}) \subseteq \textsc{Reach}_{\infty}(\mathcal{R}, \textsc{sc})$, then the smaller set of states $\mathcal{R}' = \textsc{Reach}_{=\alpha}(\textsc{Reach}_{=\delta}(x, \textsc{cc}), \textsc{sc}) \subseteq \mathcal{R}$ will also satisfy the condition $\textsc{Reach}(\mathcal{R}', \textsc{sc}) \cap U = \emptyset$. Therefore, every state reached after $\delta + \alpha$ is also admissible.

Since all three cases do not contain an inadmissible state, our assumption that an inadmissible state is reached is violated, yielding a contradiction. Therefore, this is a safe switching condition. □

In summary, the proposed approach is as follows: when the system is well inside the ellipsoid that represents the largest safe sublevel set of the Lyapunov function, we do not need to invoke an extensive reachability analysis using the safety controller, as we know the state is recoverable (even for the next control period). When the system state is near the boundary of the ellipsoid, the reachability analysis is used to allow the system to cross the boundary of the ellipsoid as long as the reachability computation shows that (1) no system constraints are violated when this is done (i.e., none of the reachable states violate a system constraint), and (2) the state can be guaranteed to be brought back into the ellipsoid (i.e., the reachable states return inside the ellipsoid). This allows the complex controller to be used in a larger region compared with the LMI approach because it can soundly reason about the behavior of the system outside of the ellipsoid (remember that the Lyapunov function from the LMI method is only a sufficient condition for safe switching). This condition can also be less conservative than the pure reachability approach because the computation needed is from a single state x, rather than the possibly large set of inadmissible states. Additionally, it involves reasoning over a finite-time horizon ($\alpha + \delta$), rather than infinite-time reachability needed in the method based on Theorem 1.

There are still two issues that need to be addressed before the condition in Theorem 4 is usable. First, since we cannot compute reachability exactly for complex hybrid automata due to decidability reasons [Henzinger et al. 1995], we will instead compute an overapproximation. This will result in a conservative switching set depending on the accuracy of the computation. Second, this computation is defined from the system's current state x, which is not available offline. In order to resolve this issue, we propose an online, real-time reachability computation method in the next section. After that, in Section 5, we will evaluate the conservatism in the switching set due to the overapproximation in the proposed algorithm.

4. REAL-TIME REACHABILITY ALGORITHM

Hybrid-systems reachability computations have been traditionally computed offline, and are both memory- and processor-intensive operations. In Section 3, we have illustrated several reasons to perform the reachability computation at runtime. This requires a reachability algorithm capable of use within a real-time system. In this section, we describe a real-time reachability algorithm with the following key features:

—High-performance for a quick runtime for short reachability times
—The ability to check the three conditions from Theorem 4
—No dynamic data structures (or large memory preallocation) or recursion, for usability in a real-time system
—No dependence on complex external libraries (only the C standard library) that most, if not all, other reachability approaches use
—Iterative improvement in accuracy with increased computation time

The last point is important because it allows the reachability task to be scheduled in the framework of imprecise real-time system computation [Liu et al. 1994]. In this framework, each task produces a partial result that is usable and improved upon as more computation time is added (this is sometimes called an *anytime algorithm*). In particular, the proposed reachability algorithm is based on the milestone approach [Lin et al. 1987], in which partial results are recorded at various points during the execution, and the last-recorded values are used when the final result is needed. This is in contrast to the traditional real-time systems execution model, in which each task has a fixed worst-case execution time (WCET) [Liu and Layland 1973].

We now present the real-time reachability algorithm that is suitable for real-time, online computation that satisfies these requirements. We distinguish between *reach time*, which is the time that we are computing reachability for, and *runtime*, which is the duration of (wall) time that the method is allowed to run. Recall that the types of hybrid systems that we consider are ones in which the state invariants are disjoint and cover the continuous state $\mathbb{R}^n$, there are no reset maps in the transitions between discrete states, and the guards of incoming transitions are defined by the state invariants. In these piecewise systems, the state of the hybrid automaton can be determined solely by the continuous state, although different differential equations can be used in different parts of the state space. This is applicable to many state-feedback continuous systems with saturation (such as those using gain scheduling controllers) since the states in which saturation occurs are typically disjoint from the unsaturated states (because the actuator command is a function of the state), and the continuous states do not jump along the saturation boundary.

To employ the real-time reachability algorithm, as in our earlier work [Bak et al. 2011], the user defines the system dynamics through a function (a function written in the C language in this implementation) that returns the minimum and maximum derivative in each dimension given an arbitrary box of the state space. The derivative needed in the algorithm is always in the outward direction of the box of states being tracked. The tracked box has $2n$ faces, where n is the number of dimensions. For each of the n dimensions, these faces are represented by a minimum face and a maximum face. That is, there are total $2n$ minimum and maximum faces, each of which refers to particular faces of a hyperrectangle used to represent portions of the set of reachable states. If the minimum face is being considered, the minimum of the derivative is used, as this may (but not necessarily so) push the tracked states outward from the hyperrectangle. If the maximum face is being considered, the maximum of the derivative is used for the same reason. Nonlinear dynamics are permitted in this approach, so long as the user-provided function maximizes or minimizes the nonlinear derivatives within an arbitrary box. Notice that this does not require solving the differential equations (which is generally a harder problem), since the bounds are on the derivatives themselves. Furthermore, we require that the derivatives are defined in the entire state space, and that they are bounded.

ALGORITHM 1: The Real-Time Reachability Algorithm Uses a Desired Reach-Time Step to Tune its Runtime Based on the Available Computation Time

```
Box currentBox := initialBox

while (reachTimeRemaining > 0)
  Box[] nebs = constructNeighborhoods(currentBox, reachTimeStep)

  crossReachTime := minCrossReachTime(nebs)
  advanceReachTime := min(crossReachTime, reachTimeRemaining)
  currentBox := advanceBox(nebs, advanceReachTime)

  reachTimeRemaining := reachTimeRemaining − reachTimeToAdvance
end while
```

The real-time reachability algorithm is based on mixed face lifting [Dang and Maler 1998; Dang 2000]. This approach is a *flow-pipe construction* method, which means that snapshots of the reachable set of states are computed at increasing points in reach-time, and reasoning is done about which states can be encountered between snapshots.

To create a real-time implementation, we use boxes (n-dimensional hyper-rectangles) as our representation of the set of states. Over long reach times, this representation can be problematic because, if the actual reachable set of states is not a box, error is introduced by overapproximating it as one (called the wrapping effect [Moore 1966]). However, since we need to compute reachability only for short reach times ($\delta + \alpha$ from Theorem 4), a simpler, faster representation is preferred for better long-term error control. In mixed face lifting, the dynamics along each face are overapproximated by the maximum derivative along that face. The reach time is then advanced uniformly along all faces (i.e., in all directions).

We modify the original mixed face-lifting algorithm to make it usable in a real-time setting. In particular, instead of using the desired error in order to control the neighborhood width around each face [Dang 2000], we use a desired reach-time step to control neighborhood widths. This parameter allows us to tune the total number of steps used in the method, and therefore alter the runtime. After the given reach time is obtained, the desired step size is decreased (which reduces the width of the neighborhoods, and therefore the derivative error at each step) and the computation is restarted. In our algorithm, initially, we use a time step that is some factor, say one-tenth, of the desired reach time. The decrease is computed by dividing the time step by two. In this way, the algorithm will produce progressively more accurate answers, for as much runtime as the task is given.

The high-level algorithm, given a fixed desired step size (`reachTimeStep`), is given in Algorithm 1. For a box, there are two faces for every dimension (one for each of the minimum and maximum faces along that dimension), and there are two corresponding face neighborhoods (regions in which the face may advance through during the current time step) for every dimension. The neighborhoods, `nebs`, are constructed based on the desired reach-time step. This neighborhood construction process will be elaborated on shortly.

Next, the minimum reach-time for any point along each face to cross the corresponding neighborhood in the corresponding direction is computed. What this means is that, for example, in the two-dimensional example of Figure 3, the minimum reach time for any point along the left face of `currentBox` to cross to the left side of `nebs[0]` in the x direction is computed, as well as the minimum reach time for any point along the right face to cross `nebs[1]`, as well as the neighborhoods in the y directions; then, the

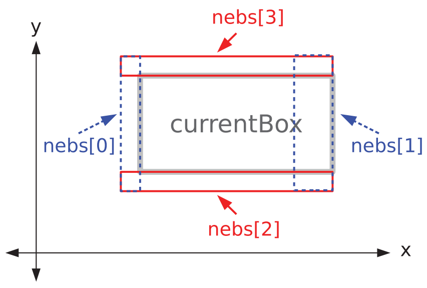

Fig. 3. The neighborhood widths are determined by `reachTimeStep` and the derivatives along the faces of `currentBox`.

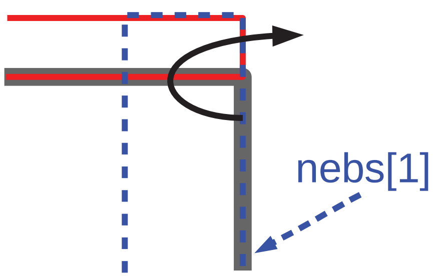

Fig. 4. Although the derivative along the face may be inward-facing, the derivative in the neighborhood can still be outward facing. The first condition of Step 3 in the neighborhood construction process checks for this and reconstructs the neighborhoods if such a situation occurs. Here, `nebs[1]` would be updated to an outward-facing neighborhood, which would require subsequent reconstruction of the other neighborhoods (because the edges overlap).

minimum of all of these is returned. This is computed by looking at the minimum or maximum derivative within the box for each neighborhood (from the user-provided derivative bounds function), as well as the width of the neighborhood along the corresponding dimension.

Finally, the `currentBox` at the next reach-time step is computed based on the neighborhoods and computed reach time to advance (which may be reduced if it exceeds `reachTimeRemaining`). This is done by advancing each face by the maximum derivative in the outward direction in its neighborhood (from the user-provided derivative bounds function) multiplied by `advanceReachTime`.

The novel aspect of this face-lifting reachability algorithm is that the widths of the neighborhoods are tunable by the `reachTimeStep` parameter. The neighborhood construction (the `constructNeighborhoods` function) proceeds in three steps:

(1) The maximum outward derivative along each face of `currentBox` is computed. One neighborhood is constructed for each face, in which the width of the corresponding neighborhood is based on the derivative (the width is the derivative multiplied by the passed-in desired `reachTimeStep`).
(2) The neighborhood boxes are all constructed based on the computed widths, such that the edges overlap as shown in Figure 3. We call a neighborhood constructed on the inside of the corresponding face an *inward-facing* neighborhood (such as `nebs[1]` in Figure 3).
(3) The outward derivatives in the constructed neighborhoods are computed with the user-provided function. If either (1) an inward-facing neighborhood contains an outward-facing derivative, or (2) a derivative has doubled in value since the previous derivative computation for that neighborhood (which initially is the flat neighborhood), the width of the neighborhood is recomputed and the process repeats by returning to Step 2.

The check in Step 3 ensures two things. The first condition is necessary in case a derivative was inward-facing in a previously constructed neighborhood, but outward-facing in the new, larger neighborhood. This case is shown in Figure 4. The second condition guarantees that the reach time to progress from a point on the face through the corresponding face neighborhood is at least `reachTimeStep`/2. Due to this, we can bound the maximum number of iterations of the while loop as the desired reach time divided by `reachTimeStep`/2. Since the edges of the neighborhoods overlap, the neighborhoods of the other faces need to be reconstructed as well, which is why the algorithm backtracks to Step 2.

The number of times that the neighborhood construction backtracks from Step 3 to Step 2 is also bounded. This is because a face can flip from inward-facing to outward-facing only once, and since it was assumed that there is a maximum derivative in the state space, the observed derivative can only double a finite number of times.

The imprecise computation version of the algorithm proceeds by running Algorithm 1 repeatedly, decreasing `reachTimeStep` after each repetition. In our implementation, after each execution, `reachTimeStep` was halved, although strategies other than halving are also possible (this is a trade-off between the time between milestones and the error reduction obtained at each iteration). When the deadline is reached (or the real-time reachability task is stopped), the most recent reachability result is used as the output. For this reason, the exact number of iterations of the neighborhood construction loop is not too useful, as long as it has an upper bound, and we can adjust it with `reachTimeStep`.

If the derivative doubles several times, the tracked box will be pessimistic, since the conservatism comes from overapproximating a derivative in a neighborhood by its maximum value. For this reason, we also set a threshold in the loop for how large the tracked boxes are allowed to get (not shown); if it is exceeded, we immediately halve `reachTimeStep` and restart the loop. If the number of backtracks to Step 2 is small (which is true in practice), each advancement of time takes $O(n)$, where n is the number of dimensions in the system.

From the four desired properties of a real-time reachability algorithm mentioned earlier, this algorithm is quick (no exponential complexity operations), requires no dynamic memory or recursion, and can iteratively provide a better answer. In order to satisfy the remaining desired condition, we need to provide the ability to check the three conditions from Theorem 4. Rather than first computing the reachable set of states and then checking the conditions in that set (which would require dynamic storage to store the reachable set), we instead modify the core algorithm in Algorithm 1 to do the checks during the computation. Conditions (1) and (2) of the theorem deal with the safety of reachable states at intermediate reach times. This can be checked inside the while loop by taking the convex hull of `currentBox` before and after the `advanceTime` call, and passing it to a function that ensures that the hull does not contain a state that violates the system constraints. For checking Condition (3), the final `currentBox` value can be used. Furthermore, these checks can be done at each iteration of the refinement; if a `reachTimeStep` is found such that the three conditions of the theorem are satisfied, no further refinement is necessary (and the complex/smart controller can be used).

5. EVALUATION

We now present an evaluation of the proposed methodology.[3] The real-time reachability approach computes the set of reachable states for the safety controller as depicted in the automaton representing the Simplex architecture in Figure 5. We demonstrate the method through two related case studies: a nonlinear inverted pendulum and a linear inverted pendulum. As another benefit of the real-time reachability method described in this article, it can also work even if the LMI approach cannot be used. We note that the LMI approach, in general, cannot be used for nonlinear systems, so its application is limited. In order to directly show the advantage of the approach in the linear case, we

[3]As it is difficult to present all the details necessary to replicate our results in an article, the source code implementation of the real-time reachability algorithm and our models are available as supplementary material and online at: http://www.verivital.com/rtreach/.

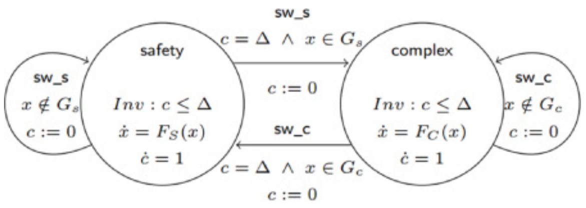

Fig. 5. Hybrid automaton for the Simplex architecture on which the real-time reachability computation is performed for the safety controller mode, for which a formal model is available. Here, Δ is a control period, c is a timer, G_S is a guard governing the transition from the safety to complex controllers, G_c is a guard governing the transition from the complex to safety controllers, and F_S and F_C denote the dynamics of the overall closed-loop system when using the safety and complex controllers. respectively.

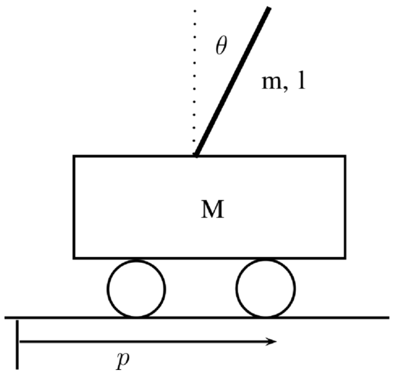

Fig. 6. An inverted pendulum system keeps a rod upright at an unstable equilibrium point by controlling a cart at its base.

use the same case study that demonstrated the earlier, LMI-based Simplex work [Seto and Sha 1999]. The linear inverted pendulum model is obtained by linearizing the nonlinear inverted pendulum model. Overall, their results are comparable and were used to validate against one another. Both models are briefly discussed here, with more details on the nonlinear and linear models in the earlier report [Seto and Sha 1999]. The system is an inverted pendulum with state constraints and input saturation. The physical system is shown in Figure 6 and consists of a DC-motor-driven cart that moves along a 1D track with a pendulum arm attached by an angular joint to the cart. The control objective is to keep the angle θ of the pendulum arm at 0° measured from the vertical (i.e., to keep the arm upright).

There are four state variables: cart position p, cart velocity $v = \dot{p}$, pendulum arm angle θ, and pendulum arm angular velocity $\omega = \dot{\theta}$. We denote x as the state vector and p as the position, seen together in Equation (1):

$$x = \begin{bmatrix} p \\ v \\ \theta \\ \omega \end{bmatrix} = \begin{bmatrix} p \\ \dot{p} \\ \theta \\ \dot{\theta} \end{bmatrix}, \text{ yielding the dynamics } \dot{x} = \begin{bmatrix} \dot{p} \\ \ddot{p} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} v \\ \dot{v} \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix}. \quad (1)$$

The system is subject to physical constraints. The range of p is between $[-1, 1]$ meters, $\dot{p}$ is between$[-1.0, 1.0]$ meters/s, θ is between $[-15, 15]^\circ$, and $\dot{\theta}$ is unconstrained (although the constraints on $\dot{p}$ do impose limits on $\dot{\theta}$). We ignore static friction (with respect to the cart wheels and ground, and with respect to the pendulum arm and joint) and take the armature inductance ($L_a = 18$ millihenries) to be 0 henries, hence reducing the order of the system by making the armature current state variable I_a a function of only V_a. Without this simplification, two control states would be necessary.

5.1. Nonlinear Inverted Pendulum

The inverted pendulum's state evolves according to a nonlinear differential equation $\dot{x} = f(x, u)$. Specifically,

$$f(x,u) = \begin{bmatrix} \dot{p} \\ \ddot{p} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} v \\ -\dfrac{\frac{1}{3}l^2\, m\,(C_1 + f_c) - \frac{1}{2} l\, m \cos\ (\theta)\ \left(C_2 + f_p\right)}{D} \\ \omega \\ -\dfrac{\bar{M}(C_2 + f_p) - \frac{1}{2} l\, m \cos\ (\theta)\ (C_1 + f_c)}{D} \end{bmatrix}. \tag{2}$$

Here, $D_l = 4\bar{M} - 3m$, $\bar{B} = \frac{K_g B_m}{r^2} + \frac{K_g^2 K_i K_b}{r^2 R_a}$, $B_l = \frac{K_g K_i}{r R_a}$, $\bar{M} = \frac{m+M+(K_g J_m)}{r^2}$, $f_c = B_x\, v + A_x\, e^{-C_x\,|v|}\ \text{sign}\ (v)$, $f_p = B_\theta\, \omega + A_\theta\, e^{-C_\theta\,|\omega|}\ \text{sign}\ (\omega)$, $D = l^2\, m(\frac{M}{3} + \frac{m}{3} + \frac{J_m\, K_g}{3\, r^2}) - \frac{l^2\, m^2 \cos\ (\theta)^2}{4}$, $C_1 = v\,(\frac{B_m\, K_g}{r^2} + \frac{K_b\, {K_g}^2\, K_i}{R_a\, r^2}) - \frac{l\, m\omega^2\ \sin\ (\theta)}{2}$, and $C_2 = -\frac{g\, l\, m \sin\ (\theta)}{2}$. The pendulum model involves the following parameters: g is gravity, R_a is the armature resistance, r is the driving wheel radius, J_m is the motor rotor inertia, B_m is the motor's coefficient of viscous friction, B_θ is the pendulum joint's coefficient of viscous friction, K_i is the motor torque constant, K_b is the motor back-e.m.f. constant, K_g is the gear ratio, M is the cart mass, m is the pendulum arm mass, l is the pendulum arm length, f_c is the static friction force, and f_p is the viscous friction force. After evaluating values for constant parameters (the same as those used in Seto and Sha [1999]), we have:

$$f(x,u) = \begin{bmatrix} \dot{p} \\ \ddot{p} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} v \\ -\frac{0.020833\omega^2 \sin(\theta) - 0.059221v + 0.25\cos(\theta)(0.0001\omega + 2.45\sin(\theta))}{0.0625\cos(\theta)^2 - 0.604167} \\ \omega \\ \frac{0.000725\omega + 17.7625\sin(\theta) - 0.25\cos(\theta)(-0.25\sin(\theta)\omega^2 + 0.710657v)}{0.0625\cos(\theta)^2 - 0.604167} \end{bmatrix}. \tag{3}$$

As illustrated in Figure 7 by the decreasing size of the overapproximation of the set of reachable states, the more runtime given to the real-time reachability algorithm, the more accurate the result (see also Figure 8 and Tables I, II, and III). These results illustrate that the real-time reachability algorithm presented in this article is effective even for hybrid systems with nonlinear differential equations. Thus, the results are widely applicable to many realistic systems.

5.2. Linearized Inverted Pendulum

As discussed in Section 5.1, generally, the system is nonlinear, $\dot{x} = f(x, u)$, but to apply the LMI-Simplex approach as a part of the unified Simplex method described in this article, we next work with a model linearized around the origin, which is the equilibrium point:

$$\dot{x} = Ax + Bu. \tag{4}$$

The linearization is justified since the control objective is to stabilize the system in a neighborhood of the vertical equilibrium, defined in this coordinate system as $\theta = 0^\circ$, which is at the origin.

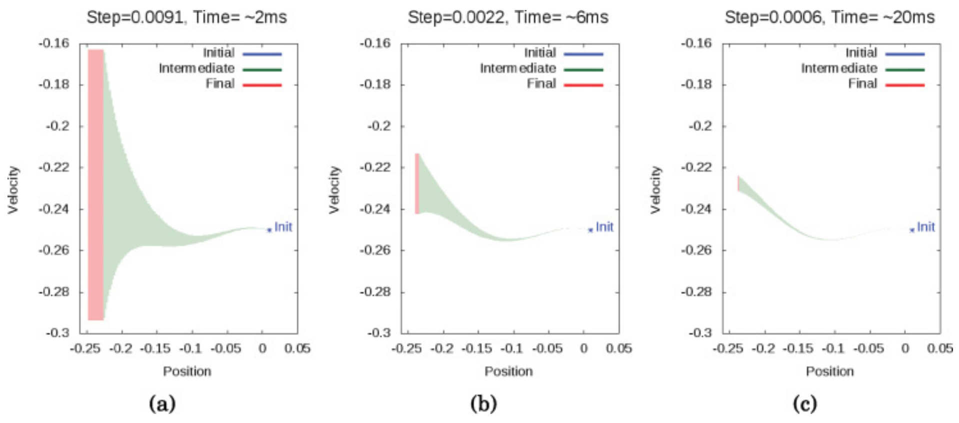

Fig. 7. Overapproximation of the set of reachable states computed by the real-time reachability method for the nonlinear inverted pendulum model Equation (2) and Equation (3) for different amounts of computation runtime, 2ms in (a), 6ms in (b), and 20ms in (c).

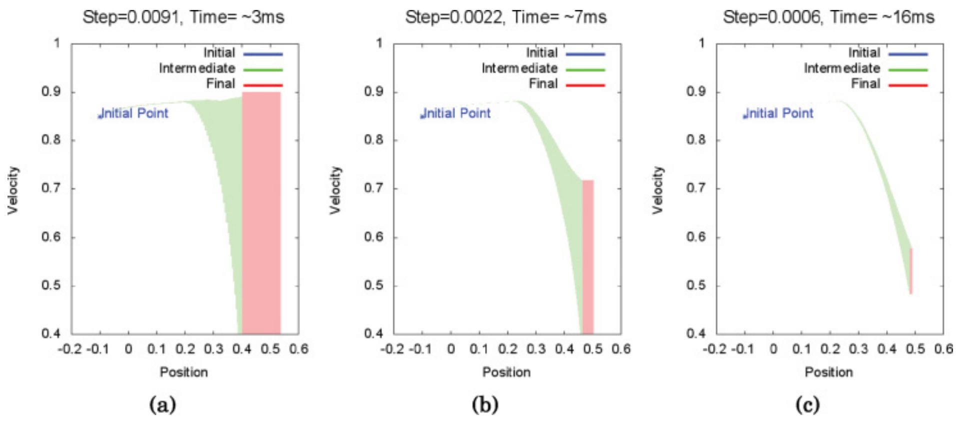

Fig. 8. Overapproximation of the set of reachable states computed by the real-time reachability method for the linearized inverted pendulum model Equation (5) and Equation (6) for different amounts of computation runtime, 3ms in (a), 7ms in (b), and 16ms in (c). The plots illustrate 2D projections of the reachable sets for the linearized inverted pendulum from the state $x = [-0.1,\ 0.85,\ 0,\ 0]^T$ for reach time 0.73. Here, the initial state is outside of the LMI-recoverable ellipsoid ($x^T P x = 1.56$), but can be proven to reenter the ellipsoid after 0.73 reach time, despite the presence of input saturation.

The plant system matrix and input vector used in Equation (4) are:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -a_{22} & -a_{23} & a_{24} \\ 0 & 0 & 0 & 1 \\ 0 & a_{42} & a_{43} & -a_{44} \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 \\ b_2 \\ 0 \\ -b_4 \end{bmatrix}, \tag{5}$$

where $a_{22} = \frac{4\bar{B}}{D_l}$, $a_{23} = \frac{3mg}{D_l}$, $a_{24} = \frac{6B_\theta}{lD_l}$, $a_{42} = \frac{6\bar{B}}{lD_l}$, $a_{43} = \frac{6\bar{M}g}{lD_l}$, $a_{44} = \frac{12\bar{M}B_\theta}{ml^2 D_l}$, $b_2 = \frac{4B_l}{D_l}$, and $b_4 = \frac{6B_1}{lD_l}$, for all the parameters defined in Section 5.1. Using the parameters from the earlier Simplex report [Seto and Sha 1999], the A and B matrices used in Equation (4) corresponding to Equation (5) are:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -10.95 & -2.75 & 0.0043 \\ 0 & 0 & 0 & 1 \\ 0 & 24.92 & 28.58 & -0.044 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 \\ 1.94 \\ 0 \\ -4.44 \end{bmatrix}. \tag{6}$$

The system is stabilized by linear state feedback of the form $\dot{x} = (A + BK)x$. The control input, $u = Kx$, is the armature voltage of a DC-motor (V_a) and is constrained between $[-4.95, 4.95]$ volts. Additionally, this control saturation prevents the system from being globally stable. The safety controller is designed following the LMI-based Simplex approach described in Section 2. The LMI approach outputs a set of gains for the safety control K, such that, when the input $u = Kx$ is used, the system will remain inside the ellipsoid also output by the method. Without saturation, the system evolves according to $\dot{x} = (A + BK)x$.

The solution to this is $x(t) = e^{(A+BK_\sigma)t}x_0$, where $x_0 \in \mathbb{R}^{4\times 1}$ is an initial condition vector. Note that, as only θ and p are observable (in the control theoretic sense, but that is, are measured by sensors), $\dot{\theta}$ and $\dot{x}$ are constructed by the first-order approximations $\dot{\theta}(t) = \frac{[\theta(t)-\theta(t-mT_s)]}{mT_s}$ and $\dot{p}(t) = \frac{[p(t)-p(t-mT_s)]}{mT_s}$, where m is an integer greater than one (chosen as 2 by experimentation). In the safety and experimental controllers, this first-order approximation is accomplished by storing a buffer of previously sampled values.

5.3. Feasible and Stabilizable Regions

Next, we discuss how to compute the feasible and stabilizable regions, defined previously in Section 2.1. We use YALMIP [Löfberg 2004], the SDPT-3 [Toh et al. 1999] solver, and MATLAB to solve the following semidefinite quadratic programming problem and underapproximate the recoverable states for the safety controller. For computing the stabilizable region for the safety controller, we find the gain vector during the optimization. The problem is to maximize the volume of the ellipsoid (thus maximize the set of recoverable states) defined by:

$$\mathcal{R} = \{x \mid x^T P x \leq 1\}. \tag{7}$$

The LMI to find the positive definite P may be formulated as:

$$\begin{gathered} \min \log \det Q^{-1} \\ \text{subject to } Q\bar{A}^T + \bar{A}^T Q < 0,\ Q > 0,\ \alpha_k^T Q \alpha_k \leq 1,\ k = 1, \ldots, n, \end{gathered}$$

where $\bar{A} = A + BK$, $Q = P^{-1}$, and the α_k for $k \in \{1, \ldots, n\}$ encode the state and control constraints. Full details of this process are given in Appendix A2 of the LMI-Simplex technical report [Seto and Sha 1999].

Variants of this process may either take a given gain vector K or find a gain vector K [Seto and Sha 1999]. For our use, the output of this process is both the gain vector K and the matrix P defining a subset of the recoverable states $\mathcal{R}$, such that, when the gain matrix is used for the safety controller, and the state is in $\mathcal{R}$, the state is guaranteed to stay in $\mathcal{R}$ indefinitely (since $V(x) = x^T P x$ is a Lyapunov function). Furthermore, all the constraints (including saturation limits) are satisfied for all states in $\mathcal{R}$. The gain vector K produced for the described pendulum system is [0.4072, 7.2373, 18.6269, 3.6725].

5.4. Comparison between Simplex with LMI and Real-Time Reachability

We now provide a comparison between control based on the $\mathcal{R}$ from the LMI approach described earlier and the switching condition produced by the proposed unified approach that uses real-time reachability. For real-time reachability, we implemented the algorithm from Section 4. In order to be usable in a real-time control system, our implementation was written in C and had no dynamic memory allocations or recursion, and used no nonstandard external libraries. In our implementation, we would call the real-time reachability C code from within MATLAB on either Linux or Windows. For the experiments described here, we first used a modern laptop with a quadcore Intel Core i7-2800MQ processor and 32GB RAM (although the computation does not require significant memory, as described earlier). Next, we evaluated the methods on embedded platforms. The first embedded platform is a BeagleBone Black development board with a 1GHz ARM processor and 512MB RAM running Debian Linux with the Xenomai real-time Linux extensions. The second embedded platform is an Arduino Yun, which has both a 400MHz MIPS processor and a 16MHz 8b Atmel AVR ATmega32u4 processor; we used the ATmega32u4 for our evaluation, in part to validate our claims on minimal resources requirements. Together, these evaluations validate our claims that the real-time reachability method is cross-platform and requires minimal processing resources. The effort to port from the original x86 implementation [Bak et al. 2014] to both the ARM and AVR implementations took about two weeks of development time, which, from a systems development standpoint, is minimal given the insurmountable difficulties that would exist in porting all the libraries required in other existing hybrid-systems reachability approaches.

One remaining input for the algorithm is the reach time necessary for a specific state to reenter $\mathcal{R}$ (the time $\delta + \alpha$ from Theorem 4). This was approximated using Euler-based simulation, which added a fixed overhead at the start of the computation. For states slightly outside of $\mathcal{R}$, the necessary reach time was typically in the hundreds of milliseconds. Since the reachability computation incurs error due to overapproximation, we compute the set of reachable states for slightly more (1.2 times) than the time the simulation took to reach $\mathcal{R}$. If the Euler simulation did not enter $\mathcal{R}$ by some upper bound (4s reach-time), the state was considered unrecoverable. A projection of the computed overapproximation of the set of reachable states for various runtimes is shown in Figure 8. As more computation runtime is added, the accuracy increases, as indicated by the size of the set decreasing.

One difference between the approaches is that the LMI-Simplex method needs to reason about one-step reachability of the plant state for any complex/smart controller command in order to compute the distance d in Figure 2. The proposed online approach, in contrast, knows what complex-controller command will be applied and can use that as part of the reachability computation. For this reason, we restrict the comparison to examine only the recoverable region for the safety controller. In this way, we do not give our approach the advantage of knowing exactly what command the complex/smart controller is using.

Our comparison shows three different approaches for estimating the recoverable region (Figures 9 and 10). First, using the LMI-only Simplex, we get a subset of the recoverable region $\mathcal{R}$. Next, using a simulation-based analysis in MATLAB, we can see an approximation of all recoverable states, which would be an ideal switching set. If the simulation returns to a steady state, then the initial point is marked as existing in the recoverable set. Finally, we show the states that the real-time reachability-based approach can guarantee as recoverable, which is somewhere between the previous two regions. For these experiments, in order to be runnable in the control loop, the runtime for the reachability code was capped at 20ms.

The stabilizable regions for p and $\dot{p}$ of the controller is seen in Figure 9, and the regions for θ and $\dot{\theta}$ of each controller are in Figure 10. One reason why the runtime reachability approach can recover more states is that the recoverable set contains states in which input saturation occurs, whereas the set $\mathcal{R}$ contains no such states.

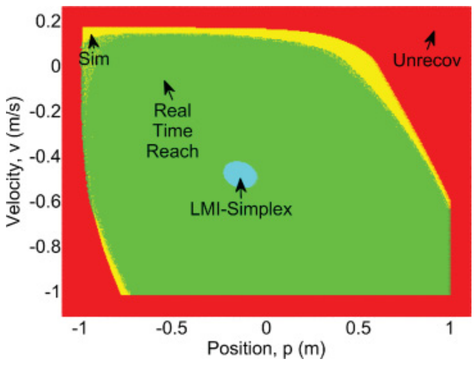

Fig. 9. Estimated projections are shown of the LMI-Simplex recoverable region $\mathcal{R}$ (cyan center set), real-time reachability recoverable region (green middle set), Simulink/Stateflow simulations that converge (yellow middle set), and simulations that diverge (red exterior set), where $\theta = 0.19$ rad ($\sim 10.89^\circ$) and $\dot{\theta} = 0.18$ rad ($\sim 10.31^\circ$) per second.

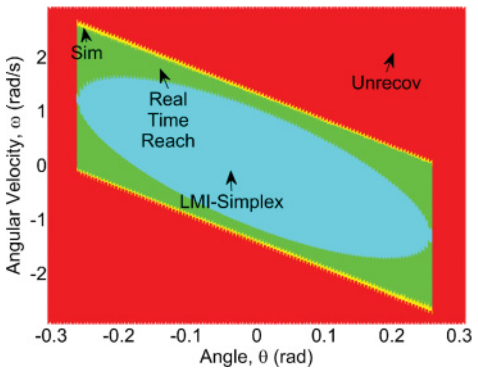

Fig. 10. Estimation of LMI-Simplex recoverable region $\mathcal{R}$ (cyan center set), real-time reachability recoverable region (green interior set), Simulink/Stateflow simulations that converge (yellow middle set), and simulations that diverge (red exterior set) shown on the projection of θ and $\dot{\theta} = \omega$ onto the $p = 0$ m and $v = 0$ m/s plane.

The largest improvements in the switching set for the real-time approach occur under this saturation situation, because reachability is able to reason about the behavior of the saturated system. Another reason for the improvement is that the LMI-produced switching set must be an ellipsoid, whereas the true set of recoverable states can be an arbitrary (even nonconvex) shape. This is seen in Figure 9, in which, since the projection is near the maximum values of θ and $\dot{\theta}$, the LMI ellipsoid projected onto this plane is small. In Figure 10, the LMI-Simplex recoverable region is clearly ellipsoidal (as expected from Equation (7)). In both Figures 9 and 10, the benefit of using real-time reachability is highlighted by the larger provably safe recoverable region. In both cases, even for a 20ms runtime, the set of states proven recoverable using real-time reachability is very close to the simulations that converge, which means that the real-time reachability is close to optimal in estimating the actual recoverable region.

Next, we evaluated the effect of varying the runtime in the real-time reachability method on the resultant switching set, which is summarized in Table I. For this table, we sampled the state-space uniformly between the state bounds presented earlier using 15 points in each dimension (thus, $15^4 = 50625$ points) in the hyper-rectangle $-1.25 \leq p \leq 1.25$ (m), $-1.2 \leq \dot{p} \leq 1.2$ (m/s), $-20 \leq \theta \leq 20$ (degrees), and $-30 \leq \dot{\theta} \leq 30$ (degrees/s). The columns LMI, Real Time, Sim, and Unrecov list the number of recoverable points for each approach (in terms of recoverable states, notice that LMI $\subseteq$ Real Time $\subseteq$ Sim), as measured by the uniform sampling. The column Recoverable is the comparison of the number of states verified safe in the proposed unified method with real-time reachability over the earlier LMI-Simplex approach. The improvement is an estimate of the increased state-space size (volume) allowed using our real-time reachability method, over using only the LMI-based recoverable region. Since the real-time recoverable states contain all the LMI-Simplex states, the improvement is calculated as: (#RealTime Points + #LMI Points)/(#LMI Points). For a runtime of 20ms, the improvement in volume of the switching set is estimated at 227%, whereas, based on simulations, we estimate the maximum possible improvement in Recoverable to be around 247% (calculated as (#Sim Points + #RealTime Points + #LMI Points)/(#LMI Points)).

We experimented with increasing the number of samples up to 30 points in each dimension, which yielded similar improvements, and in the limit as the number of

Table I. Intel Core i7 x86-64: PC Evaluation Summary of Experiments Varying Runtime

Runtime (ms)	LMI	Real Time	Sim	Unrecov	Recoverable
5	5473	5971	14323	24858	209%
20	5473	6753	13541	24858	223%
40	5473	6974	13320	24858	227%
50	5473	7081	13213	24858	229%
75	5473	7109	13185	24858	230%
100	5473	7183	13111	24858	231%
200	5473	7273	13021	24858	233%
500	5473	7338	12956	24858	234%
1000	5473	7382	12912	24858	235%
2000	5473	7424	12870	24858	236%
3000	5473	7428	12866	24858	236%
4500	5473	7448	12846	24858	236%
6000	5473	7455	12839	24858	236%

samples tends to infinity, we would converge to the exact improvement. However, these approximations are reasonable based on the consistency of our experimental results (e.g., 20ms runtime for 15 samples is about a 227% improvement, and it is also about a 230% improvement for 30 samples). As expected, as the runtime allowed for real-time reachability increases, the improvement increases since the real-time reachability implementation uses an anytime approach and refines the precision of the reachability computation based on available runtime. Even for small runtimes (e.g., 5ms), the improvement is already significant at over 200% more provably recoverable states, which makes the approach promising for implementation in real-time control loops.

5.5. Comparison on ARM and Arduino AVR ATmega32u4 Embedded Hardware Platforms

Next, we compare the benefit of using our real-time reachability approach versus the LMI-Simplex method on actual embedded hardware platforms. The first hardware platform is an ARM processor in the TI Sitara system-on-chip used in the CircuitCo BeagleBone Black development kit. The specific ARM processor is an AM335x 1GHz ARM Cortex-A8 with the NEON floating-point accelerator and access to 512MB DDR3 RAM. The experiments were conducted on a Debian Linux distribution with a kernel modification to use the Xenomai real-time Linux extensions, enabling use of real-time operating system (RTOS) features within Linux. A summary of experimental results are reported in Table II. Here, we can see that for reasonable runtimes, even on an embedded platform (tens of milliseconds), the approach presented in this article has an improvement of around 1.5 to 2 times over the LMI approach. For runtimes on the order of hundreds of milliseconds to seconds, the approach yields similar improvements to the desktop implementation. For this table (as with Table I), we sampled the state-space uniformly between the state bounds presented earlier using 15 points in each dimension (thus, $15^4 = 50625$ points) in the same hyper-rectangle used in the earlier experiment, specifically, $-1.25 \leq p \leq 1.25$ (m), $-1.2 \leq \dot{p} \leq 1.2$ (m/s), $-20 \leq \theta \leq 20$ (degrees), and $-30 \leq \dot{\theta} \leq 30$ (degrees/s).

The second hardware platform is the Arduino Yun. The Yun has both a 400MHz MIPS processor and a 16MHz 8b Atmel AVR ATmega32u4. For this evaluation, we use the 16MHz ATmega32u4 processor, which is representative of small, memory constrained embedded devices. The ATmega32u4 has available 2.5KB SRAM, 32KB of flash memory, but because the real-time reachability method does not use any dynamic memory allocation and does not rely on any nonstandard libraries, we are able to run

Table II. BeagleBone Black ARM: Embedded System Evaluation Summary of Experiments Varying Runtime

Runtime (ms)	LMI	Real Time	Sim	Unrecov	Recoverable
5	5473	2270	18024	24858	141%
20	5473	3832	16462	24858	170%
40	5473	4613	15681	24858	184%
50	5473	4617	15677	24858	184%
75	5473	5350	14944	24858	198%
100	5473	5361	14933	24858	199%
200	5473	5968	14326	24858	209%
500	5473	6721	13573	24858	223%
1000	5473	6952	13342	24858	227%
2000	5473	7107	13187	24858	230%
3000	5473	7110	13184	24858	230%
4500	5473	7216	13078	24858	232%
6000	5473	7216	13078	24858	232%

Table III. Arduino Atmel AVR ATmega32u4: Embedded System Evaluation Summary of Experiments Varying Runtime

Runtime (ms)	LMI	Real Time	Sim	Unrecov	Recoverable
100	2088	0	8226	10422	100%
500	2088	192	8034	10422	109%
1000	2088	566	7660	10422	127%
2000	2088	879	7347	10422	142%
3000	2088	882	7344	10422	142%
4500	2088	1198	7028	10422	157%

it on the platform in spite of the processing and memory constraints. Although the implementation runs with the restricted resources, the runtime is noticeably higher than on the other processors. In this case, the system would only stand to benefit if the dynamics were sufficiently slow (thus, a runtime of seconds would be tolerable) or if we further optimized parts of the implementation for the limited resources (changing software floating-point computations to use fixed-point, since there is no FPU on the ATmega32u4. A summary of experimental results for the AVR are reported in Table III. For this table (unlike in Tables I and II), we sampled the state-space uniformly between the state bounds presented earlier using 12 points in each dimension (thus, $12^4 = 20736$ points) in the same hyper-rectangle as the earlier experiments, specifically, $-1.25 \le p \le 1.25$ (m), $-1.2 \le \dot{p} \le 1.2$ (m/s), $-20 \le \theta \le 20$ (degrees), and $-30 \le \dot{\theta} \le 30$ (degrees/s). While the AVR is too resource constrained to be able to improve the states usable in the control period time (20ms) and requires on the order of hundreds of milliseconds to seconds to yield an improvement, this is, to the best of our knowledge, the first demonstration of a reachability method in a resource-constrained embedded system of this scale. We also highlight that simply performing a simulation on the AVR requires about hundreds of milliseconds of runtime.

6. RELATED WORK

The Simplex architecture [Sha 2001; Seto and Sha 1999] has been used extensively to provide guarantees for systems that use untrusted logic. It has been used for systems ranging from offroad vehicles [Bak 2009], to models of airplanes [Seto et al. 2000], to fleets of remotely controlled cars [Crenshaw et al. 2007], to networked control systems [Yao et al. 2013]. Recently, variants of Simplex have been proposed to account for physical-system (plant) failures [Wang et al. 2013], faults in the OS or microprocessor

[Bak et al. 2009], and to check for security intrusions [Mohan et al. 2013]. Simplex is closely related to Run-Time Assurance (RTA) methods [Clark et al. 2013; Murthy 2012]. RTA methods were used to construct a safe supervisory control system for a simulation of a high-altitude unmanned aerial vehicle [Aiello et al. 2010]. Here, a transition function that projects the current state to a future state was used to determine the switching boundary. This transition function, as well as the recoverable states, were determined through extensive simulation and online prediction of trajectories. The proposed real-time reachability approach in this work could be used to provide verified bounds on the transition function used in RTA methods. This work also mentions the interesting idea of using an online/offline design for switching module logic by leveraging a simplified model of the plant dynamics and taking the model error into account when doing the switching, which could reduce the complexity of the online reachability computation.

Earlier work has also integrated traditionally nonreal-time search approaches within real-time systems [Musliner and Durfee 1995]. In this approach, AI planning techniques were discussed in the context of real-time systems, and two categories of possible integration were proposed: (1) the nonreal-time algorithms were adapted to run in a real-time fashion or (2) they were run in a supervisory mode, not as part of the real-time control loop. Real-time reachability would fall in the former category in this classification.

A related notion to Simplex in control theory is that of a viability kernel [Aubin 1991]. A viability kernel is a set of states in which there exists a trajectory that stays within a predefined environment. Viability kernels can be approximated for linear systems, for example, by using analysis of random directions in the state space [Gillula et al. 2014]. Reachability analysis of hybrid systems has also been extensively researched in the last 20 years [Guéguen et al. 2009]. Reachability analysis tools exist for classes of systems with timed [Bengtsson et al. 1996], rectangular [Henzinger et al. 1997; Frehse 2008; Johnson and Mitra 2014], linear [Frehse et al. 2011; Frehse 2008], and nonlinear [Ratschan and She 2007; Tiwari 2008; Bak 2013a; Chen et al. 2012; Benvenuti et al. 2014; Duggirala et al. 2015] dynamics, with varying degrees of accuracy and scalability. Other bounded model checking (BMC) tools for hybrid systems built on SMT solvers also exist [Eggers et al. 2011; Gao et al. 2013]. However, to the best of our knowledge, the algorithms in earlier reachability and BMC tools were all designed for offline analysis, not for real-time, in-the-loop computation. Specifically, real-time reachability requires performance to be predictable, which is difficult to ensure when there are large external libraries, huge code bases, and significant use of dynamic memory. For example, one popular reachability analysis tool for affine hybrid automata is SpaceEx, which requires at least eight external libraries: Parma Polyhedra Library (PPL) [Bagnara et al. 2008], Boost C++ Libraries, GNU Multiple Precision Arithmetic Library (gmplib), GNU Linear Programming Kit (glpk), SUNDIALS (Solver Suite) [Hindmarsh et al. 2005], aaflib, ublasJama, and TinyXML [Frehse et al. 2011].[4] Another recent tool, C2E2, relies on at least 11 external libraries: GNU Linear Programming Kit (glpk and pyglpk), GNU Parser Generator (Bison), Fast Lexical Analysis (FLEX), Python, Python Parsing Libraries (Python-PLY), GTK Libraries for Python (PyGTK), Plotting Libraries for Python (Matplotlib), Packing Configurations Library (pkg-config), GNU Autoconf (autoconf), Python XML Library (lxml), and Parma Polyhedral Library (PPL).[5] While several of these libraries would not need to be executed in the reachability computation (such as those related to parsing and package management), several libraries (PPL, gmplib, glpk, and SUNDIALS) are fundamental to the reachability computations. For

[4]http://spaceex.imag.fr/licensing-45.
[5]https://publish.illinois.edu/c2e2-tool/download/.

these core libraries, it would be essentially impossible to convert SpaceEx or C2E2 to a real-time implementation, as several of these libraries are incredibly complex (specifically, PPL, gmplib, glpk, and SUNDIALS).

The real-time reachability approach described in this article primarily solves the problem of computing the *continuous successors* in a hybrid automaton, although it can also be applied to invariant-disjoint hybrid dynamics. Research in computing continuous successors is related to validated integration, which traditionally has been done using interval analysis [Moore 1966], as well as intervals with preconditioning to reduce wrapping-effect error [Stauning 1997]. More recently, Taylor models have also been proposed as an alternative shown to provide superior long-term error control [Neher et al. 2007], which has been integrated into a more full hybrid automaton model checker [Chen et al. 2012]. However, the challenge for runtime approaches such as the one proposed in this article is more with quick computation of reasonable accuracy rather than long-term error control, and we are unaware of any previous real-time validated integration approaches.

Some recent work performs online reachability computation with existing, nonreal-time algorithms. This can be used, for example, when systems do not have statically known models [Bu et al. 2011]. This work, however, treats the reachability computation as a black box, which may or may not complete (because it does not use a real-time reachability algorithm). Another work also uses existing reachability approaches such as PHAVer [Frehse 2008] in a medical safeguard system [Li et al. 2012], resulting in a system that may add safety, but only if the computation completes on time. While a theoretical upper bound on execution time may be formulated due to decidability of the particular class of hybrid automata considered [Li et al. 2014], the implementation of PHAVer does not provide such guarantees, and it is not clear that such a bound would be usable or too pessimistic. A real-time reachability algorithm that always provides an answer like our approach could be integrated into both of these approaches.

Finally, the results of formal approaches are only as good as the model that they are provided. Accurate system identification [Söderström and Stoica 1988] is therefore essential. The approach here reduces pessimism in the switching logic *for a given model*. Accuracy and validation of the model itself is an important problem, but beyond the scope of this work. Recent approaches from the hybrid-systems community, however, have begun to make use of runtime monitors to do online checking of model accuracy [Mitsch and Platzer 2014].

7. CONCLUSION AND FUTURE WORK

In this work, we have proposed an alternate unified design for Simplex that leverages two existing design methodologies based on control-theoretic LMI optimization and hybrid-systems reachability. Our unified approach extends the region where the complex/smart controller enabling smart autonomy can be used by leveraging a real-time reachability computation, thus decreases conservatism in the switching logic. Using a runtime of 20ms (which matches the control loop period time), we were able to expand the set of states in which the complex/smart controller could be used by 227%, whereas we estimated, through simulation, that the maximum improvement possible was approximately 247%. Even with a reduced real-time reachability runtime of 5ms, we were able to improve upon the LMI-based Simplex design by 213%. On embedded processors, we were also able to increase the complex/smart controller region by a factor of 1.5 to 2.0; for an 8b microcontroller, however, the current implementation was not fast enough for use at the frequency of the control loop. This improvement was demonstrated in an evaluation that uses the exact system previously used to demonstrate the LMI-based Simplex design approach, an inverted pendulum with input saturation.

The real-time reachability computation is able to predict the behavior of the system despite saturation, significantly expanding the usable complex/smart controller region.

To the best of the authors' knowledge, this is the first work to present a viable real-time reachability algorithm based on the real-time systems notion of imprecise computation. The algorithm will always return an overapproximation of the set of reachable states, with better accuracy as more computation time is given. The key difference between online reachability compared with offline reachability, besides constrained runtime and resources, is that quick results are preferable to long-term error control. In our evaluation, for example, we were able to demonstrate significant improvement in the complex/smart controller region by using tens of milliseconds of computation time to bound the future behavior of the system for the next hundreds of milliseconds. Together, our evaluation on actual embedded hardware platforms, including ARM processors and Atmel AVR microcontrollers, illustrates the embedded usage feasibility of using the real-time reachability method. Other reachability algorithms also contain parameters that could be tuned to have some control over the computation time, such as the sampling time used in the Le Geurnic-Girard (LGG) scenario in SpaceEx [Frehse et al. 2011]; we plan to investigate better approaches for real-time reachability.

Real-time reachability has applications beyond just determining Simplex switching logic, however. We foresee future applications involving online system identification, detecting sensor spoofing, runtime verification, and enabling a variant of model-predictive control (MPC). To enable these applications, we are implementing code generation capabilities in the HYST model transformation and translation tool for hybrid automata [Bak et al. 2015], which will enable creating implementations of the real-time reachability algorithm for large classes of hybrid automata.

ACKNOWLEDGMENTS

The authors are grateful for the feedback from the anonymous reviewers of this article and its earlier revision [Bak et al. 2014] that significantly helped improve this article. The material presented in this article is based upon work supported by the National Science Foundation (NSF) under grant numbers CNS-1302563, CNS-1219064, CNS 13-29886, CNS 13-30077, CNS 1464311, and CCF 1527398, by the Office of Naval Research (ONR) under grant number N00014-12-1-0046, by the Air Force Research Laboratory's Information Directorate (AFRL/RI) through the Visiting Faculty Research Program (VFRP) under contract number FA8750-13-2-0115, AFRL through contract number FA8750-15-1-0105, the Air Force Office of Scientific Research (AFOSR) in part under contract number FA9550-15-1-0258, and the AFOSR Summer Faculty Fellowship Program (SFFP). The U.S. government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the AFRL, AFRL/RI, AFOSR, NSF, or ONR.

REFERENCES

Michael Aiello, John Berryman, Jonathan Grohs, and John Schierman. 2010. Run-time assurance for advanced flight-critical control systems. In *Proceedings of the American Institute of Aeronautics and Astronautics Guidance, Navigation, and Control Conference (AIAA'10)*.

R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. 1995. The algorithmic analysis of hybrid systems. *Theoretical Computer Science* 138, 3–34.

Rajeev Alur and David L. Dill. 1994. A theory of timed automata. *Theoretical Computer Science* 126, 183–235.

Jean-Pierre Aubin. 1991. *Viability Theory*. Birkhauser Boston Inc., Cambridge, MA.

R. Bagnara, P. M. Hill, and E. Zaffanella. 2008. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming* 72, 1–2, 3–21.

Stanley Bak. 2009. *Industrial Application of the System-Level Simplex Architecture for Real-Time Embedded System Safety*. Master's thesis. University of Illinois at Urbana-Champaign, Champaign, IL.

Stanley Bak. 2013a. HyCreate: A Tool for Overapproximating Reachability of Hybrid Automata. Retrieved January 17, 2016 from http://stanleybak.com/projects/hycreate/hycreate.html.

Stanley Bak. 2013b. *Verifiable COTS-Based Cyber-Physical Systems*. Ph.D. dissertation. University of Illinois at Urbana-Champaign, Urbana, IL.

Stanley Bak, Sergiy Bogomolov, and Taylor T. Johnson. 2015. HyST: A source transformation and translation tool for hybrid automaton models. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control (HSCC'15)*. ACM, New York, NY.

Stanley Bak, Deepti K. Chivukula, Olugbemiga Adekunle, Mu Sun, Marco Caccamo, and Lui Sha. 2009. The system-level Simplex architecture for improved real-time embedded system safety. In *15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'09)*.

Stanley Bak, Ashley Greer, and Sayan Mitra. 2010. Hybrid cyberphysical system verification with Simplex using discrete abstractions. In *IEEE Real-Time and Embedded Technology and Applications Symposium,* Vol. 0. IEEE Computer Society, Los Alamitos, CA, 143–152. DOI:http://dx.doi.org/10.1109/RTAS.2010.27

Stanley Bak, Taylor T. Johnson, Marco Caccamo, and Lui Sha. 2014. Real-time reachability for verified Simplex design. In *IEEE Real-Time Systems Symposium (RTSS'14)*. IEEE Computer Society, Rome, Italy.

Stanley Bak, Karthik Manamcheri, Sayan Mitra, and Marco Caccamo. 2011. Sandboxing controllers for cyber-physical systems. In *Proceedings of International Conference on Cyber-Physical Systems (ICCPS'11)*.

Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. 1996. UPPAAL: A tool suite for automatic verification of real-time systems. In *Hybrid Systems III*, Rajeev Alur, Thomas Henzinger, and Eduardo Sontag (Eds.). Lecture Notes in Computer Science, Vol. 1066. Springer, Berlin, 232–243. DOI:http://dx.doi.org/10.1007/BFb0020949

Luca Benvenuti, Davide Bresolin, Pieter Collins, Alberto Ferrari, Luca Geretti, and Tiziano Villa. 2014. Assume-guarantee verification of nonlinear hybrid systems with Ariadne. *International Journal of Robust and Nonlinear Control* 24, 4, 699–724. DOI:http://dx.doi.org/10.1002/rnc.2914

S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. 1994. *Linear Matrix Inequalities in System and Control Theory*. Studies in Applied Mathematics, Vol. 15. SIAM, Philadelphia, PA.

M. S. Branicky. 1998. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control* 43, 4, 475–482. DOI:http://dx.doi.org/10.1109/9.664150

Lei Bu, Qixin Wang, Xin Chen, Linzhang Wang, Tian Zhang, Jianhua Zhao, and Xuandong Li. 2011. Toward online hybrid systems model checking of cyber-physical systems' time-bounded short-run behavior. *SIGBED Rev* 8, 2, 7–10.

Xin Chen, Erika Abraham, and Sriram Sankaranarayanan. 2012. Taylor model flowpipe construction for non-linear hybrid systems. *2013 IEEE 34th Real-Time Systems Symposium* 0, 183–192. DOI:http://dx.doi.org/10.1109/RTSS.2012.70

Matthew Clark, Xenofon Koutsoukos, Ratnesh Kumar, Insup Lee, George Pappas, Lee Pike, Joseph Porter, and Oleg Sokolsky. 2013. Study on Run Time Assurance for Complex Cyber Physical Systems. Technical Report, Air Force Research Lab, Wright-Patterson AFB, OH.

Tanya L. Crenshaw, Elsa Gunter, C. L. Robinson, Lui Sha, and P. R. Kumar. 2007. The Simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures. In *RTSS'07*. Washington, DC, 400–412. DOI:http://dx.doi.org/10.1109/RTSS.2007.50

Thao Dang. 2000. *Verification et Synthese des Systemes Hybrides*. Ph.D. Dissertation. INPG, Grenoble, France.

Thao Dang and Oded Maler. 1998. Reachability analysis via face lifting. In *Hybrid Systems: Computation and Control (HSCC'98)*. Lecture Notes in Computer Science, Vol. 1386. Springer, Berlin, 96–109.

Thao Dang, Oded Maler, and Romain Testylier. 2010. Accurate hybridization of nonlinear systems. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC'10)*. ACM, New York, NY, USA, 11–20.

Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. 2015. C2E2: A verification tool for stateflow models. In *Tools and Algorithms for the Construction and Analysis of Systems*, Christel Baier and Cesare Tinelli (Eds.). Lecture Notes in Computer Science, Vol. 9035. Springer, Berlin, 68–82. DOI:http://dx.doi.org/10.1007/978-3-662-46681-0_5

Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. 2013. Verification of annotated models from executions. In *Proceedings of the 11th ACM International Conference on Embedded Software (EMSOFT'13)*. IEEE Press, Piscataway, NJ, Article 26, 10 pages.

Andreas Eggers, Nacim Ramdani, Nedialko Nedialkov, and Martin Fränzle. 2011. Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In *Software Engineering and*

Formal Methods, Gilles Barthe, Alberto Pardo, and Gerardo Schneider (Eds.). Lecture Notes in Computer Science, Vol. 7041. Springer Berlin, 172–187. DOI:http://dx.doi.org/10.1007/978-3-642-24690-6_13

Goran Frehse. 2008. PHAVer: Algorithmic verification of hybrid systems past HyTech. *International Journal on Software Tools for Technology Transfer (STTT)* 10, 3, 263–279. DOI:http://dx.doi.org/10.1007/s10009-007-0062-x

Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification (CAV)*. Lecture Notes in Computer Science. Springer, Berlin.

Sicun Gao, Soonho Kong, and Edmund Clarke. 2013. Satisfiability modulo ODEs. In *International Conference on Formal Methods in Computer-Aided Design (FMCAD'13)*. DOI:http://dx.doi.org/10.1109/FMCAD.2008.ECP.14

Jeremy H. Gillula, Shahab Kaynama, and Claire J. Tomlin. 2014. Sampling-based approximation of the viability kernel for high-dimensional linear sampled-data systems. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control (HSCC'14)*. ACM, New York, NY, 173–182. DOI:http://dx.doi.org/10.1145/2562059.2562117

Hervé Guéguen, Marie-Anne Lefebvre, Janan Zaytoon, and Othman Nasri. 2009. Safety verification and reachability analysis for hybrid systems. *Annual Reviews in Control* 33, 1, 25–36. DOI:http://dx.doi.org/DOI:10.1016/j.arcontrol.2009.03.002

Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. 1997. HyTech: A model checker for hybrid systems. *Journal on Software Tools for Technology Transfer* 1, 1, 110–122. DOI:http://dx.doi.org/10.1007/s100090050008

Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. 1995. What's decidable about hybrid automata? In *Journal of Computer and System Sciences*. ACM Press, New York, NY, 373–382.

Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. 2005. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software* 31, 3, 363–396. DOI:http://dx.doi.org/10.1145/1089014.1089020

Taylor T. Johnson and Sayan Mitra. 2014. Anonymized reachability of rectangular hybrid automata networks. In *Formal Modeling and Analysis of Timed Systems (FORMATS'14)*.

J. Kapinski and B. H. Krogh. 2002. A new tool for verifying computer controlled systems. In *IEEE Conference on Computer-Aided Control System Design*. 98–103.

H. K. Khalil. 2002. *Nonlinear Systems* (3rd ed.). Prentice Hall, Upper Saddle River, NJ.

Gerardo Lafferriere, George J. Pappas, and Shankar Sastry. 2000. O-minimal hybrid systems. *Mathematics of Control, Signals and Systems* 13, 1, 1–21.

Tao Li, Feng Tan, Qixin Wang, Lei Bu, Jian-Nong Cao, and Xue Liu. 2012. From offline toward real-time: A hybrid systems model checking and CPS co-design approach for medical device plug-and-play (MDPnP). In *2012 IEEE/ACM 3rd International Conference on Cyber-Physical Systems (ICCPS'12)*. 13–22. DOI:http://dx.doi.org/10.1109/ICCPS.2012.10

Tao Li, Feng Tan, Qixin Wang, Lei Bu, Jian-Nong Cao, and Xue Liu. 2014. From offline toward real time: A hybrid systems model checking and CPS codesign approach for medical device plug-and-play collaborations. *IEEE Transactions on Parallel and Distributed Systems* 25, 3, 642–652. DOI:http://dx.doi.org/10.1109/TPDS.2013.50

Kwei-Jay Lin, Swaminathan Natarajan, and Jane W.-S. Liu. 1987. Imprecise results: Utilizing partial computations in real-time systems. In *RTSS*. 210–217.

C. L. Liu and J. W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery* 20, 1.

J. W. S. Liu, Wei-Kuan Shih, Kwei-Jay Lin, R. Bettati, and J. Y. Chung. 1994. Imprecise computations. *Proceedings of the IEEE* 82, 1, 83–94. DOI:http://dx.doi.org/10.1109/5.259428

J. Löfberg. 2004. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*. Taipei, Taiwan. http://users.isy.liu.se/johanl/yalmip/.

Stefan Mitsch and Andre Platzer. 2014. ModelPlex: Verified runtime validation of verified cyber-physical system models. In *Runtime Verification*, Borzoo Bonakdarpour and Scott A. Smolka (Eds.). Lecture Notes in Computer Science, Vol. 8734. Springer, Berlin, 199–214. DOI:http://dx.doi.org/10.1007/978-3-319-11164-3_17

Sibin Mohan, Stanley Bak, Emiliano Betti, Heechul Yun, Lui Sha, and Marco Caccamo. 2013. S3A: Secure system Simplex architecture for enhanced security and robustness of cyber-physical systems. In *Proceedings of the 2nd ACM International Conference on High Confidence Networked Systems (HiCoNS'13)*. 10. DOI:http://dx.doi.org/10.1145/2461446.2461456

R. E. Moore. 1966. *Interval Analysis*. Prentice-Hall.

Abhishek Murthy. 2012. Simplex Architecture for Run Time Assurance of Hybrid Systems. *Safe and Secure Systems and Software Symposium* (S5).

David J. Musliner and Edmund H. Durfee. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74, 1, 83–127. DOI:http://dx.doi.org/10.1016/0004-3702(94)00008-O

M. Neher, K. R. Jackson, and N. S. Nedialkov. 2007. On Taylor model based integration of ODEs. *SIAM Journal on Numerical Analysis* 45.

Stefan Ratschan and Zhikun She. 2007. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Transactions on Embedded Computing Systems* 6, 1, Article 8. DOI:http://dx.doi.org/10.1145/1210268.1210276

Danbing Seto, Enrique Ferreira, and Theodore F. Marz. 2000. *Case Study: Development of a Baseline Controller for Automatic Landing of an F-16 Aircraft using Linear Matrix Inequalities (LMIs)*. Carnegie Mellon University Software Engineering Institute, Pittsburgh, PA 15213. Technical report number CMU/SEI-99-TR-020. http://www.sei.cmu.edu/reports/99tr020.pdf.

D. Seto and Lui Sha. 1999. *A Case Study on Analytical Analysis of the Inverted Pendulum Real-Time Control System*. CMU/SEI Technical Report 99-TR-023. Carnegie Mellon University, Pittsburgh, PA.

Lui Sha. 2001. Using simplicity to control complexity. *IEEE Software* 18, 4, 20–28. DOI:http://dx.doi.org/dx.doi.org/10.1109/ MS.2001.936213

T. Söderström and P. Stoica (Eds.). 1988. *System Identification*. Prentice-Hall, Inc., Upper Saddle River, NJ.

O. Stauning. 1997. *Automatic Validation of Numerical Solutions*. Ph.D. Dissertation. Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby.

Ashish Tiwari. 2008. Abstractions for hybrid systems. *Formal Methods in System Design* 32, 1, 57–83. DOI:http://dx.doi.org/10.1007/s10703-007-0044-3

K. C. Toh, M. J. Todd, and R. H. Tutuncu. 1999. SDPT3: A MATLAB software package for semidefinite programming. *Optimization Methods and Software* 11, 545–581.

Lieven Vandenberghe, Stephen Boyd, and Shao-Po Wu. 1998. Determinant maximization with linear matrix inequality constraints. *SIAM Journal on Matrix Analysis and Applications* 19, 2, 499–533. DOI:http://dx.doi.org/10.1137/S0895479896303430

Xiaofeng Wang, N. Hovakimyan, and Lui Sha. 2013. L1Simplex: Fault-tolerant control of cyber-physical systems. In *2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS'13)*. 41–50. DOI:http://dx.doi.org/10.1109/ICCPS.2013.6603998

Jianguo Yao, Xue Liu, Guchuan Zhu, and Lui Sha. 2013. NetSimplex: Controller fault tolerance architecture in networked control systems. *IEEE Transactions on Industrial Informatics* 9, 1, 346–356. DOI:http://dx.doi.org/10.1109/TII.2012.2219060

Received December 2014; revised June 2015; accepted September 2015