Real-Time Reachability for Verified Simplex Design

Stanley Bak[†], Taylor T. Johnson[‡], Marco Caccamo[*], Lui Sha[*]

[†]United States Air Force Research Lab - Information Directorate
[‡]University of Texas at Arlington
[*]University of Illinois at Urbana-Champaign

***Abstract*—The Simplex Architecture ensures the safe use of an unverifiable complex controller by using a verified safety controller and verified switching logic. This architecture enables the safe use of high-performance, untrusted, and complex control algorithms without requiring them to be formally verified. Simplex incorporates a supervisory controller and safety controller that will take over control if the unverified logic misbehaves. The supervisory controller should (1) guarantee the system never enters and unsafe state (safety), but (2) use the complex controller as much as possible (minimize conservatism).**

The problem of precisely and correctly defining this switching logic has previously been considered either using a control-theoretic optimization approach, or through an offline hybrid systems reachability computation. In this work, we prove that a combined online/offline approach, which uses aspects of the two earlier methods along with a real-time reachability computation, also maintains safety, but with significantly less conservatism. We demonstrate the advantages of this unified approach on a saturated inverted pendulum system, where the usable region of attraction is 227% larger than the earlier approach.

I. Introduction

Modern cyber-physical systems are large complex systems of systems, where arguments about the behavior of the whole system rely on guarantees about the individual components. Individual components, however, may be designed using machine learning methods such as neural networks that are currently not amenable to formal analysis, or the components may simply be too large and complex for complete verification.

One approach to provide formally verified behavior despite the use of unverified control logic is the Simplex Architecture [1]. 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 controller) is wrapped with a verified controller (the safety controller) and switching logic (the decision module). The complex 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 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.

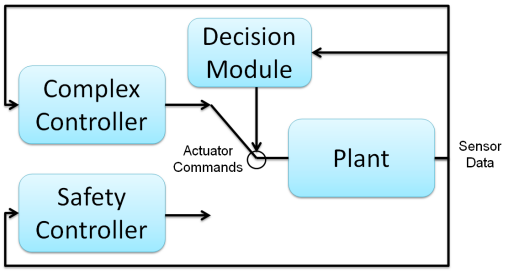

Figure 1. The Simplex Architecture produces a verified system despite the use of an unverified complex controller. The decision module should switch between the controllers to provide overall system safety.

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 controller is never used. The key challenge, which is the focus of this paper, 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 safely recover the system.

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 [2]. Alternatively, approaches based on hybrid systems reachability can be used to produce a provably safe decision module [3]. These earlier approaches will be reviewed in Section II. In this paper, we propose the use of a unified approach, where the offline LMI result is combined with an online reachability computation to produce a *significantly less conservative* Simplex system which is still safe. We elaborate on this approach and prove its safety in Section III.

DISTRIBUTION A. Approved for public release; Distribution unlimited. (Approval AFRL PA case number 88ABW-2014-4185, 03 SEP 2014)

The proposed approach requires computing reachability online for short time intervals. Previous hybrid systems reachability algorithms, however, were not designed for real-time computation. For this reason, in Section IV, we propose a real-time reachability algorithm based on mixed face lifting [4] which is compatible with the imprecise computation model in real-time scheduling literature [5]. 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 V. 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 run-time approach significantly expands the space where the complex controller may be used.

Other work related to Simplex and reachability is then presented in Section VI, followed by conclusions in Section VII.

II. Background

There have been several verified design methodologies for systems which 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 which cannot be violated, or linearization boundaries where the model is considered valid.

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

From this definition, we can then 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. 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.

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

A. Verified Design using LMI

The first proposed way to design a verified decision module is based on solving linear matrix inequalities [2], [6], which has been used to design Simplex systems as complicated as automated landing maneuvers for an F-16 [7]. 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, as well as 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 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 ($x^T Px$) which is positive-definite with negative-definite derivative (so it is monotonically decreasing over time), thus guaranteeing stability of the linear system using Lyapunov's direct or indirect (if the plant is nonlinear and was linearized) methods. Furthermore, the matrix P is constructed by the method such that it defines an ellipsoid in the state space where all the constraints are satisfied when $x^T Px < 1$. Since the states where saturation occurs were input as constraints to the method, this means that states inside the ellipsoid result in control commands that are not beyond the actuator limits (where saturation would occur).

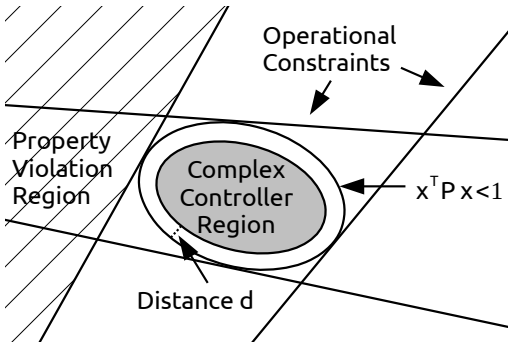

Figure 2. The LMI Simplex design approach uses switching logic based on an ellipsoid within the system constraints in order to produce a verified system.

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 visually in Figure 2.

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 controller can be used. If the state approaches the boundary of the ellipsoid, control can be switched to the safety controller which will drive the system towards the equilibrium point where $x^T Px = 0$.

Care must be taken to ensure 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 region must be used to define the region where the complex controller is allowed to actuate the system. In the figure, the distance d defines this extra buffer, which can be determined offline by computing the maximum gradient for any control command inside the ellipsoid, multiplied by the period of the decision logic.

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 controller region (as shown in Figure 2) is reentered, before switching back to the complex controller.

B. Verified Design using Reachability

An alternative method for verified Simplex design is based on reachability analysis of hybrid systems [8], which has been used, for example, to create a Simplex system to prevent off-road vehicle rollover [9]. 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 is a tuple, $H = (\mathcal{X}, L, X_0, I, F, T)$ [10], 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 possible for each location. It is a function $L \to 2^{\mathcal{X}}$.
- F is a set of flows that defines the differential equations in each location. It is a function $X \to 2^{\mathbb{R}^n}$.
- T is a set of discrete transitions that 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 $l \in L$. The guard conditions on the outgoing transitions define when the location can change, whereas the invariants of the locations can be used to force transitions by preventing time from elapsing further in the current mode. This therefore allows nondeterminsim in the discrete behavior. A hybrid automaton can be visually depicted as a finite-state machine with differential equations in each discrete state. The model also allows for nondeterminism in the continuous behavior because a single state $x \in X$ defines, via the set of flows F, a set of derivative values for each variable.

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 [11]. Thus, analysis of hybrid systems often restricts either the continuous dynamics or the discrete dynamics [12], [13], [14]. In this paper, the reachability algorithm proposed in Section IV considers restricted hybrid automata models where the state invariants are disjoint and cover the continuous states $\mathbb{R}^n$, there are no reset maps in the transitions between discrete states, and the guards of incoming transitions are defined by the state invariants.

In addition to dynamics restrictions, practical reachability approaches often over-approximate the reachable set of states [15], [16], [17], which is sufficient for proving safety properties. If a sound over-approximation of the reachable set of states of 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. This approach may, however, lead to false positives where the error in the over-approximation contains unsafe states, but the actual reachable set of states does not.

Here, $\textsc{Reach}_{\infty}(x, \textsc{ha})$ refers to all the states reached in any amount of time from state x in hybrid automaton HA, $\textsc{Reach}_{\leq t}(x, \textsc{ha})$ refers to the states reached from x in up to t time, and $\textsc{Reach}_{=t}(x, \textsc{ha})$ are the states reached after exactly t time has elapsed. Also, we naturally extend REACH to initial *sets* of states, where 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. Optimal here means that the given safety controller takes over only if it has to; if it did not take over, then the system could enter an inadmissible state in the future. Furthermore, it never takes over when the complex 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 controller modes is defined using the following theorem [3]:

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

The inner $\textsc{Reach}_{=\delta}$ in part (2) is the time-bounded reachability of the system for one decision logic switching interval time, δ, while using the complex controller (CC). The outer $\textsc{Reach}_{\infty}$ 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 controller is used for a single control interval of time δ, and 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 controller cannot be used for one more control interval, and instead the safety controller must be used right away. Assuming the system starts in a recoverable state, this guarantees 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 controller is not available, but rather an over-approximation must be used which 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. A safe switching set, however, can still be computed using over-approximations, where the conservativeness of the resultant decision module depends on the amount of over-approximation. 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 [3], [8]. As with the LMI approach, the output is a set of states which forms a guaranteed subset of the recoverable states.

III. Unified Approach for Simplex Design

The two approaches for Simplex design previously discussed each have their own limitations.

The LMI approach works with purely linear physical dynamics. 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 where 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 $Kx \leq \text{MAX\_INPUT}$ and $Kx \geq \text{MAX\_INPUT}$).

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 which, due to convexity and quadratic restrictions required in the optimization algorithms, is always an ellipsoid. This is a sufficient but not necessary condition for stability and therefore the switching set is almost certainly conservative. We demonstrate this pessimism in our evaluation in Section V.

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[1]. One downside of this approach is that over-approximation error occurs from the need to abstract the complex controller hybrid automaton by a hybrid automaton which takes into account any possible complex 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 non-convex set in many dimensions. Lastly, hybrid systems reachability over-approximation methods introduce error, especially 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 be large.

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 2. *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 $\mathcal{R} = \{x | x^T P x < 1\}$, 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 $X^T P X < 1$. Furthermore, when the gain vector K output by the approach is used (which defines the safety controller $u = Kx$), 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:

Theorem 3. *A safe switching condition for Simplex is given when, at every control iteration, the complex controller is used if, for some α time, (1)* $\text{REACH}_{\leq\delta}(x, \text{CC}) \cap U = \emptyset$*, (2)* $\text{REACH}_{\leq\alpha}(\text{REACH}_{=\delta}(x, \text{CC}), \text{SC}) \cap U = \emptyset$ *and (3)* $\text{REACH}_{=\alpha}(\text{REACH}_{=\delta}(x, \text{CC}), \text{SC}) \subseteq \mathcal{R}$.

Proof: Intuitively, this switching condition says the complex controller can be used if, (1) the complex 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 $\delta + \alpha$ times passes, and (3) after $\delta + \alpha$ time, a state in $\mathcal{R}$ will be safely reached.

More formally, assume by contradiction that this is not a safe switching condition, so 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 2 $\text{REACH}_\infty(\mathcal{R}, \text{SC}) \cap U = \emptyset$. Since if $\mathcal{R}' \subseteq \mathcal{R}$, $\text{REACH}_\infty(\mathcal{R}', \text{SC}) \subseteq \text{REACH}_\infty(\mathcal{R}, \text{SC})$, the smaller set of states $\mathcal{R}' = \text{REACH}_{=\alpha}(\text{REACH}_{=\delta}(x, \text{CC}), \text{SC}) \subseteq \mathcal{R}$ will also satisfy the condition $\text{REACH}(\mathcal{R}', \text{SC}) \cap U = \emptyset$. 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 was wrong, and 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 Lyapunov function, we do not need to invoke an extensive reachability analysis using the safety controller, as we know the state is recoverable. When the system is about to reach the boundary of the ellipsoid, runtime reachability analysis is used to allow the system to cross the boundary of the ellipsoid as long as the analysis shows that (1) no system constraints are violated when this is done, and (2) the state can be guaranteed to be brought back into the ellipsoid. This can allow 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 which need to be addressed before the condition in Theorem 3 is usable. First, since we cannot compute reachability exactly for complex hybrid automata [11], we will instead compute an over-approximation. This will result in a more conservative switching set depending

[1]Over-approximating reachability approaches typically have an accuracy / computation time trade off.

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 V, we will evaluate the conservatism in the switching set due to the over-approximation in the proposed algorithm.

IV. Real-Time Reachability Algorithm

Hybrid systems reachability computations have been traditionally computed offline, and are both memory and processor intensive operations. Above, we have demonstrated a need to perform the 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 3.
- No dynamic data structures (or large preallocations) or recursion, for usability in a real-time system.
- 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 [18]. 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 [5], where 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 where each task has a fixed worst-case execution time (WCET) [19].

We now present a reachability algorithm suitable for real-time, online, computation which satisfies the above requirements. We distinguish between *reach-time*, which is the time we are computing reachability for, and *runtime*, which is the wall time the method is allowed to run. Recall that the types of hybrid systems we consider are ones where 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 since the states where 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 proposed algorithm, as in our earlier work [3], the user defines the system dynamics through a function (a C function, in this implementation) that returns the minimum or maximum derivative in each dimension given an arbitrary box of the state space. Nonlinear dynamics are permitted in this approach, so long as the user-provided function maximizes/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 the derivatives are defined in the entire state space, and that they are bounded.

Algorithm 1 The real-time face-lifting reachability algorithm uses a desired reach-time step to tune its runtime.

```
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 proposed reachability algorithm is based on mixed face lifting [20], [21]. 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 fast implementation, we use boxes 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 over-approximating it as one (called the wrapping effect [22]). However, since we only need to compute reachability for short reach-times ($\delta + \alpha$ from Theorem 3), a simpler, faster, representation is preferred to better long-term error control. In mixed face lifting, the dynamics along each face are over-approximated by the maximum derivative along that face. Reach-time is then advanced uniformly along all faces.

We modify the original mixed face lifting algorithm to make it more usable in a real-time setting. In particular, instead of using the desired error in order to control the width neighborhood around each face [21], we instead 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 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 Figure 1. For a box, there are two faces for every dimension (one for the minimum face along that dimension and one for the maximum face),

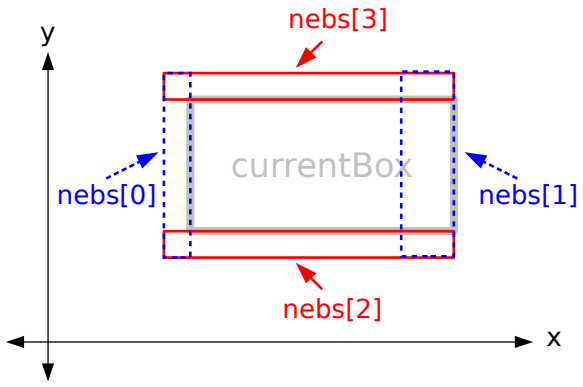

Figure 3. The neighborhood widths are determined by `reachTimeStep` and the derivatives along the faces of `currentBox`.

and thus there are also two face neighborhoods for every dimension. The neighborhoods, `nebs`, are constructed based on the desired reach-time step. This neighborhood construction process will be elaborated on later.

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 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, and then the 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 new aspect of this 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, where 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 4. We call a neighborhood constructed on the inside of the corresponding face an *inward-facing* neighborhood (such as `nebs[1]` in the figure).
3) The outward derivatives in the constructed neighborhoods are sampled. 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, the width of the neighborhood is recomputed and the process repeats by returning to step 2.

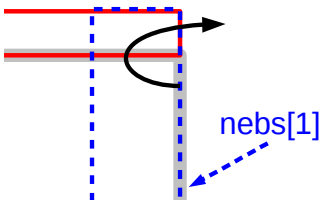

Figure 4. Although the derivative along the face may be inward-facing, the derivative in the neighborhood can still be outward facing. The first condition of step 3 in the neighborhood construction process checks for this and reconstructs the neighborhoods if such a situation occurs. Here, `nebs[1]`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

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 visually in Figure 4. The second condition guarantees that the reach-time to progress through the face 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 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 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 task is stopped), the most-recent result is 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 over-approximating 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), and 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 3. 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 Figure 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 that to a function which ensures the hull does not contain a state which 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 controller can be used).

V. Evaluation

We now present an evaluation of the proposed methodology. In order to directly show the advantage of the approach, we use the same case study which demonstrated the earlier, LMI-based Simplex work. This model is briefly discussed here, with more details in the earlier report [2]. The system is an inverted pendulum with state constraints and input saturation. The physical system is shown in Figure V and consists of a DC-motor driven cart that moves along a 1-d 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).

While the system is in general nonlinear, $\dot{x} = f(x, u)$, we work with a model linearized about the origin:

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

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

$$x = \begin{bmatrix} p \\ v \\ \theta \\ \omega \end{bmatrix} = \begin{bmatrix} p \\ \dot{p} \\ \theta \\ \dot{\theta} \end{bmatrix}. \tag{1}$$

The plant system matrix and input vector are:

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

where $a_{22} = \frac{4\bar{B}}{D_l}$, $a_{23} = \frac{3mg}{D_l}$, $a_{24} = \frac{6B_\theta}{lD_l}$, $a_{42} = \frac{6\bar{B}}{lD_l}$, $a_{43} = \frac{6\bar{M}g}{lD_l}$, $a_{44} = \frac{12\bar{M}B_\theta}{ml^2D_l}$, $b_2 = \frac{4B_l}{D_l}$, and $b_4 = \frac{6B_l}{lD_l}$, for $D_l = 4\bar{M} - 3m$, $\bar{B} = \frac{K_gB_m}{r^2} + \frac{K_g^2K_iK_b}{r^2R_a}$, $B_l = \frac{K_gK_i}{rR_a}$, $\bar{M} = \frac{m+M+(K_gJ_m)}{r^2}$, and where 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 mass, and l is the pendulum length. Using the parameters from the earlier Simplex report [2], the A and B matrices used are:

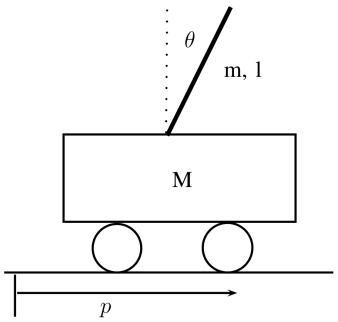

Figure 5. An inverted pendulum system maintains an upright rod by controlling a cart at its base.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -10.95 & -2.75 & 0.0043 \\ 0 & 0 & 0 & 1 \\ 0 & 24.92 & 28.58 & -0.044 \end{bmatrix}, \; B = \begin{bmatrix} 0 \\ 1.94 \\ 0 \\ -4.44 \end{bmatrix}.$$

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/second, θ is between $[-15, 15]^\circ$, and $\dot{\theta}$ is unconstrained (although the constraints on $\dot{p}$ do impose limits on $\dot{\theta}$).

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. Thus, due to this control constraint, it is necessary to look at the system in the form of $\dot{x} = Ax + Bu$ at some points of the later analysis. 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 II. 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 $\dot{x} = (A + BK)\, x$.

A. Feasible and Stabilizable Regions

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. For linear systems with constraints, this region may be under-approximated by solving an LMI of the determinant maximization form [23]. This has the effect of, for a matrix which describes an ellipsoid $x^TPx = 1$, maximizing the product of the radii of the ellipsoid (which

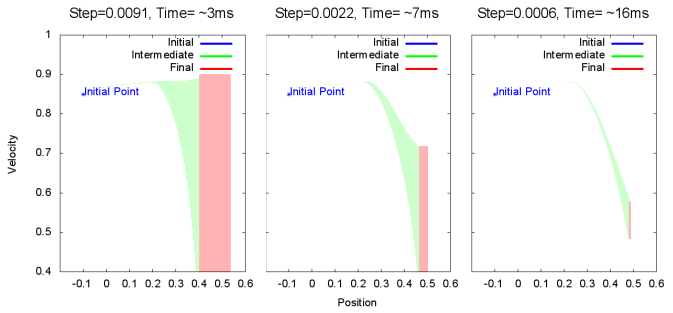

Figure 6. The more runtime given to the real-time reachability algorithm, the more accurate the result (see also Table I). Shown here are 2-d projections of the reachable sets for the inverted pendulum system from state $x = [-0.1,\ 0.85,\ 0,\ 0]^T$ for reach-time 0.73. Here, the initial state is outside 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.

is related to the determinant of the matrix P). The volume of an ellipsoid, then, is proportional to this product. We use YALMIP [24], the SDPT-3 [25] solver, and Matlab to solve the following semidefinite quadratic programming problem and under-approximate 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 (and thus maximize the set of recoverable states) defined by:

$$\mathcal{R} = \{x \mid x^T P x \leq 1\}. \tag{2}$$

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

$$\begin{aligned} &\min \log\det Q^{-1} \\ &\text{s.t.} QA^T + AQ < 0,\ Q > 0,\ \alpha_k^T Q a_k \leq 1,\ k = 1, \ldots, n, \end{aligned}$$

where $Q = P^{-1}$ and the α_k for $k \in \{1, ..., n\}$ encode the state and control constraints. More details of this process are Appendix A2 of the LMI-Simplex technical report [2].

Variants of this process may either take a given gain vector K or find a gain vector K [2]. 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 $\mathcal{R}$.

B. Comparison

We now provide a comparison between control based on the $\mathcal{R}$ from the LMI approach above, and the switching condition produced by the proposed unified approach which uses real-time reachability. For real-time reachability, we implemented the algorithm from Section IV. 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 and Windows. For the experiments, we

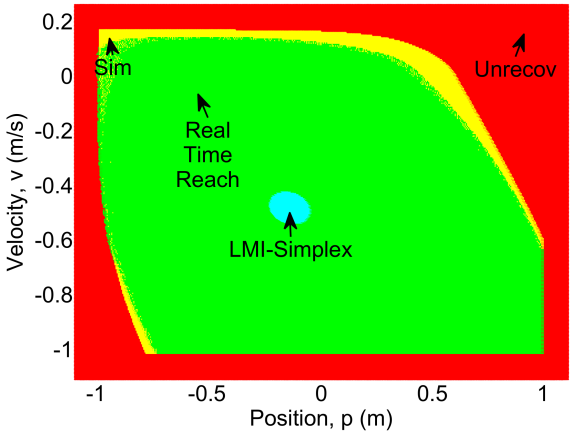

Figure 7. Estimated projections are shown of the LMI-Simplex recoverable region $\mathcal{R}$ (cyan center set), real-time reachability recoverable region (green middle set), Simulink/Stateflow simulations that converge (yellow middle set), and simulations that diverge (red exterior set) where $\theta = 0.19$ rad (~$10.89°$) and $\dot{\theta} = 0.18$ rad (~$10.31°$) per second. The LMI-Simplex ellipsoid is small here because it is a projection near the maximum values of θ and $\dot{\theta}$, although the true set of recoverable states is significantly larger.

used a modern laptop with an Intel Core i7-2800MQ processor and 32GB RAM (although the computation does not require significant memory as described earlier). 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 3). 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 reachability computation incurs error, we actually computed the reach set for slightly more (1.2 times) than what it took the simulation to reach $\mathcal{R}$. If the Euler simulation did not enter $\mathcal{R}$ by some upper bound (4 seconds reach-time), the state was considered unrecoverable. A projection of the computed reachability for various runtimes is shown in Figure 6. As more computation time is added, the accuracy increases.

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 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 only examine 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 controller is using.

Our comparison shows three different approaches for estimating the recoverable region (Figures 7 and 8). 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 the 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

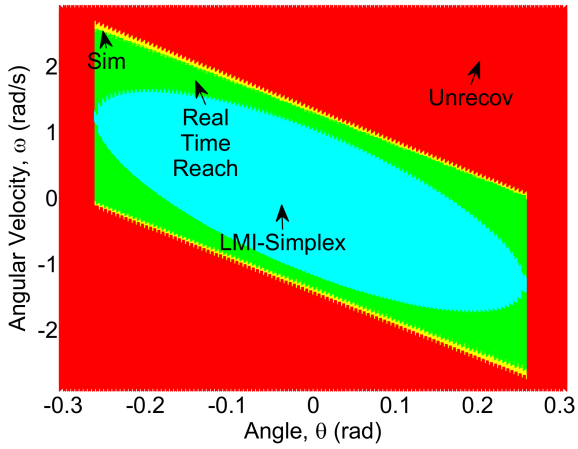

Figure 8. Estimation of LMI-Simplex recoverable region $\mathcal{R}$ (cyan center set), real-time reachability recoverable region (green interior set), Simulink/Stateflow simulations that converge (yellow middle set), and simulations that diverge (red exterior set) shown on the projection of θ and $\dot{\theta} = \omega$ onto the $p = 0$ m and $v = 0$ m/s plane.

regions. For these experiments, in order to be runnable in the control loop, the runtime for the reachability code was capped at 20 ms.

The stabilizable regions for p and $\dot{p}$ of the controller is seen in Figure 7 and the regions for θ and $\dot{\theta}$ of each controller are in Figure 8. One reason why the runtime reachability approach can recover more states is that the recoverable set contains states where input saturation occurs, whereas the set $\mathcal{R}$ contains no such states. 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 shape. This is seen in Figure 7, where, since the projection is near the maximum values of θ and $\dot{\theta}$, the LMI ellipsoid projected onto this plane is small. In Figure 8 the LMI-Simplex recoverable region is clearly ellipsoidal (as expected from Equation 2). In both Figures 7 and 8, the benefit of using real-time reachability is highlighted by the larger provably safe recoverable region. In both cases, even for a 20 ms 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 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 (so $15^4 = 50625$ points). 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 Improve is the improvement of proposed unified method with real-time reachability over the earlier LMI-Simplex approach. The improvement is an estimate of the

Runtime (ms)	LMI	RealTime	Sim	Unrecov	Improve
5	5473	6180	1376	37596	213%
20	5473	6948	608	37596	227%
40	5473	7059	497	37596	229%
50	5473	7108	448	37596	230%
75	5473	7133	423	37596	230%
100	5473	7216	340	37596	232%
200	5473	7286	270	37596	233%
500	5473	7338	218	37596	234%
1000	5473	7382	174	37596	235%

Table I
SUMMARY OF EXPERIMENTS VARYING RUNTIME

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: $(\#\text{RealTime Points}+\#\text{LMI Points})/(\#\text{LMI Points})$. For a runtime of 20 ms, the improvement in volume of the switching set is estimated at 227%, whereas based on simulations we estimate the maximum possible improvement to be around 247% (calculated as $(\#\text{Sim Points} + \#\text{RealTime Points} + \#\text{LMI Points})/(\#\text{LMI Points})$).

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 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., 20 ms 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. 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.

VI. Related Work

The Simplex Architecture [1], [2] has been used extensively to provide guarantees for systems that use untrusted logic. It has been used for systems ranging from off-road vehicles [26], to models of airplanes [7], to fleets of remotely controlled cars [27]. Recently, variants of Simplex have been proposed to account for physical-system (plant) failures [28], faults in the OS or microprocessor [29], and to check for security intrusions [30]. Simplex is closely related to Run-Time Assurance (RTA) methods [31], [32]. RTA methods were used to construct a safe supervisory control system for a simulation of a high-altitude unmanned aerial vehicle [33]. 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.

Another related notion in control theory is that of a viability kernel [34]. A viability kernel is a set of states where 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 [35]. Reachability analysis of hybrid systems has also been extensively researched in the last 20 years [36]. Reachability analysis tools exist for classes of systems with timed [37], rectangular [38], [39], linear [17], [39], and non-linear [40], [41] dynamics, with varying degrees of accuracy and scalability. However, to the best of our knowledge, the algorithms in earlier reachability tools were all designed for offline analysis, and not for real-time, in-the-loop computation. Specifically, real-time reachability requires performance to be predictable, which is difficult when there are large external libraries, huge code bases, and lots of dynamic memory.

The real-time face lifting approach described here primarily solves the problem of computing the *continuous successors* in a hybrid automaton, although as shown it can also work for invariant-disjoint hybrid dynamics. Research in computing continuous successors is related to validated integration, which traditionally has been done using interval analysis [22], as well as intervals with preconditioning to reduce wrapping-effect error [42]. More recently, Taylor models have also been proposed as an alternative shown to provide superior long-term error control [43], and this is has been integrated into a more full hybrid automaton model checker [41]. However, the challenge for runtime approaches such as the one proposed in this paper 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, non-real-time algorithms. This can be used, for example, when systems do not have statically-known models [44]. 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 [39] in a medical safeguard system [45], and results in a system which 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 [46], 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 they are provided. Accurate system identification [47] 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.

VII. Conclusion and Future Work

In this work, we have proposed an alternate design for Simplex that leverages two existing design methodologies based on control-theoretic LMI optimization and hybrid systems reachability. The unified approach extends the region where the complex controller can be used by leveraging a real-time reachability computation, and thus decreases conservatism in the switching logic. Using a runtime of 20ms (which matches the control loop iteration time), we were able to expand the set of states where the complex controller could be used by 227%, whereas we estimated, through simulation, that the maximum improvement possible was 247%. Even with a reduced real-time reachability runtime of 5ms, we were able to improve upon the LMI-based Simplex design by 213%.

As far as we know, 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 over-approximation of the reachable set, 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 controller region by using tens of milliseconds of computation time to bound the future behavior of the system for the next hundreds of milliseconds. Other reachability algorithms also contain parameters which 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 [17], and we plan to investigate better approaches for real-time reachability.

In this paper we proposed a single reachability algorithm for one class of hybrid automata. As with offline reachability computation, we believe it is necessary to develop classes of reachability algorithms that can work at runtime for various classes of hybrid automata. Real-time reachability has applications beyond just determining Simplex switching logic, however. We foresee future applications involving online system identification, detecting sensor spoofing, and enabling a variant of model-predictive control (MPC).

Acknowledgement

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant numbers CNS-1302563, CNS-1219064, CNS 13-29886, and CNS 13-30077, by the Office of Naval Research (ONR) under grant number N00014-12-1-0046, as well as the Air Force Research Laboratory's Information Directorate (AFRL/RI) through the Visiting Faculty Research Program (VFRP) under contract number FA8750-13-2-0115 and the Air Force Office of Scientific Research (AFOSR). 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 NSF, ONR, AFRL/RI, or AFOSR.

REFERENCES

[1] L. Sha, "Using simplicity to control complexity," *IEEE Softw.*, vol. 18, no. 4, pp. 20–28, 2001.

[2] D. Seto and L. Sha, "A case study on analytical analysis of the inverted pendulum real-time control system," CMU/ SEI, Tech. Rep., 1999.

[3] S. Bak, K. Manamcheri, S. Mitra, and M. Caccamo, "Sandboxing controllers for cyber-physical systems," in *Proceedings of International Conference on Cyber-physical systems (ICCPS 2011)*, 2011.

[4] T. Dang, "Verification et synthese des systemes hybrides," Ph.D. dissertation, INPG, Oct. 2000.

[5] K.-J. Lin, S. Natarajan, and J. W.-S. Liu, "Imprecise results: Utilizing partial comptuations in real-time systems," in *RTSS*, 1987, pp. 210–217.

[6] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*, ser. Studies in Applied Mathematics. Philadelphia, PA: SIAM, June 1994, vol. 15.

[7] D. Seto, E. Ferreira, and T. F. Marz, "Case study: Development of a baseline controller for automatic landing of an f-16 aircraft using linear matrix inequalities (lmis)," Technical Report Cmu/ sei-99-Tr-020. [Online]. Available: citeseer.ist.psu.edu/ 606539.html

[8] S. Bak, "Verifiable cots-based cyber-physical systems," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA, May 2013.

[9] S. Bak, A. Greer, and S. Mitra, "Hybrid cyberphysical system verification with simplex using discrete abstractions," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, ser. RTAS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 143–152.

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

[11] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" in *Journal of Computer and System Sciences*. ACM Press, 1995, pp. 373–382.

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

[13] G. Lafferriere, G. J. Pappas, and S. Sastry, "O-minimal hybrid systems," *Mathematics of Control, Signals and Systems*, vol. 13, no. 1, pp. 1–21, 2000.

[14] M. Branicky, "Multiple lyapunov functions and other analysis tools for switched and hybrid systems," *Automatic Control, IEEE Transactions on*, vol. 43, no. 4, pp. 475–482, Apr 1998.

[15] J. Kapinski and B. H. Krogh, "A new tool for verifying computer controlled systems," in *IEEE Conference on Computer-Aided Control System Design*, pp. 98–103.

[16] T. Dang, O. Maler, and R. Testylier, "Accurate hybridization of nonlinear systems," in *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, ser. HSCC '10. New York, NY, USA: ACM, 2010, pp. 11–20.

[17] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV)*, ser. LNCS. Springer, 2011.

[18] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J. Y. Chung, "Imprecise computations," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 83–94, Jan 1994.

[19] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, 1973.

[20] T. Dang and O. Maler, "Reachability analysis via face lifting," in *Hybrid Systems: Computation and Control HSCC'98*. Springer, 1998, pp. 96–109, lNCS 1386.

[21] T. Dang, "Verification et synthese des systemes hybrides," Ph.D. dissertation, INPG, oct 2000.

[22] R. Moore, *Interval analysis*, ser. Prentice-Hall series in automatic computation. Prentice-Hall, 1966.

[23] L. Vandenberghe, S. Boyd, and S.-P. Wu, "Determinant maximization with linear matrix inequality constraints," *SIAM J. Matrix Anal. Appl.*, vol. 19, no. 2, pp. 499–533, 1998.

[24] J. Löfberg, "Yalmip : A toolbox for modeling and optimization in MATLAB," in *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004. [Online]. Available: http://control.ee.ethz.ch/ joloef/yalmip.php

[25] K. C. Toh, M. J. Todd, and R. H. Tutuncu, "Sdpt3: a matlab software package for semidefinite programming," *Optimization Methods and Software*, vol. 11, pp. 545–581, 1999.

[26] S. Bak, "Industrial application of the System-Level Simplex Architecture for real-time embedded system safety," Master's Thesis, University of Illinois at Urbana-Champaign, 2009.

[27] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. R. Kumar, "The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures," in *Real-Time Systems Symposium (RTSS)*, 2007.

[28] X. Wang, N. Hovakimyan, and L. Sha, "L1simplex: Fault-tolerant control of cyber-physical systems," in *Cyber-Physical Systems (ICCPS), 2013 ACM/IEEE International Conference on*, April 2013, pp. 41–50.

[29] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "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)*.

[30] S. Mohan, S. Bak, E. Betti, H. Yun, L. Sha, and M. Caccamo, "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*, ser. HiCoNS '13, 2013. [Online]. Available: http://doi.acm.org/10.1145/2461446.2461456

[31] M. Clark, X. Koutsoukos, R. Kumar, I. Lee, G. Pappas, L. Pike, J. Porter, and O. Sokolsky, "Study on run time assurance for complex cyber physical systems," Technical Report, Air Force Research Lab, Wright-Patterson AFB, 2013.

[32] A. Murthy, "Simplex architecture for run time assurance of hybrid systems," Safe and Secure Systems and Software Symposium (S5), 2012.

[33] M. Aiello, J. Berryman, J. Grohs, and J. Schierman, "Run-time assurance for advanced flight-critical control systems," in *Proceedings of the American Institute of Aeronautics and Astronautics Guidance, Navigation, and Control Conference*, ser. AIAA '10, 2010.

[34] J.-P. Aubin, *Viability Theory*. Cambridge, MA, USA: Birkhauser Boston Inc., 1991.

[35] J. H. Gillula, S. Kaynama, and C. J. Tomlin, "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*, ser. HSCC '14, 2014, pp. 173–182.

[36] H. Guéguen, M.-A. Lefebvre, J. Zaytoon, and O. Nasri, "Safety verification and reachability analysis for hybrid systems," *Annual Reviews in Control*, vol. 33, no. 1, pp. 25–36, 2009.

[37] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL: A tool suite for automatic verification of real-time systems," in *Hybrid Systems III*, ser. LNCS, R. Alur, T. Henzinger, and E. Sontag, Eds. Springer, 1996, vol. 1066, pp. 232–243.

[38] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HyTech: A model checker for hybrid systems," *Journal on Software Tools for Technology Transfer*, vol. 1, pp. 110–122, 1997.

[39] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 10, pp. 263–279, 2008.

[40] S. Bak, "Hycreate: A tool for overapproximating reachability of hybrid automata," Demo and Poster Session, ACM/IEEE 16th International Conference on Hybrid Systems: Computation and Control (HSCC 2013), 2012. [Online]. Available: http://stanleybak.com/projects/hycreate/hycreate.html

[41] X. Chen, E. Abraham, and S. Sankaranarayanan, "Taylor model flowpipe construction for non-linear hybrid systems," *2013 IEEE 34th Real-Time Systems Symposium*, vol. 0, pp. 183–192, 2012.

[42] O. Stauning, "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, 1997.

[43] M. Neher, K. R. Jackson, and N. S. Nedialkov, "On taylor model based integration of odes," *SIAM J. Numer. Anal*, vol. 45, p. 2007.

[44] L. Bu, Q. Wang, X. Chen, L. Wang, T. Zhang, J. Zhao, and X. Li, "Toward online hybrid systems model checking of cyber-physical systems' time-bounded short-run behavior," *SIGBED Rev.*, vol. 8, no. 2, pp. 7–10, Jun. 2011.

[45] T. Li, F. Tan, Q. Wang, L. Bu, J.-N. Cao, and X. Liu, "From offline toward real-time: A hybrid systems model checking and cps codesign approach for medical device plug-and-play (mdpnp)," in *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems (ICCPS)*, April 2012, pp. 13–22.

[46] ——, "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*, 2014.

[47] T. Söderström and P. Stoica, Eds., *System Identification*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.