Periodically-Scheduled Controller Analysis using Hybrid Systems Reachability and Continuization

Stanley Bak
Air Force Research Lab - Information Directorate
Rome, NY, USA

Taylor T. Johnson
University of Texas at Arlington
Arlington, TX, USA

***Abstract*—Cyber-physical systems (CPS) consist of physical entities that obey dynamical laws and interact with software components. A typical CPS implementation includes a discrete controller, where software periodically samples physical state and produces actuation commands according to a real-time schedule. Such a hybrid system can be modeled formally as a hybrid automaton. However, reachability tools to verify specifications for hybrid automata do not perform well on such periodically-scheduled models. This is due to a combination of the large number of discrete jumps and the nondeterminism of the exact controller start time. In this paper, we demonstrate this problem and propose a solution, which is a validated abstraction mechanism where every behavior of the original sampled system is contained in the behaviors of a purely continuous system with an additive nondeterministic input. Reachability tools for hybrid automata can better handle such systems. We further improve the analysis by considering local analysis domains. We automate the proposed technique in the Hyst model transformation tool, and demonstrate its effectiveness in a case study analyzing the design of a yaw-damper for a jet aircraft.**

I. Introduction

Periodic real-time scheduling is a widespread method used to control a physical plant as part of a cyber-physical system (CPS). Typical schedulers, such as rate-monotonic (RM) or earliest deadline first (EDF) [1], give a guarantee of *periodic* execution. In each period, sensors are read, the control algorithm is run, and actuator outputs are set. The physical world, on the other hand, evolves continuously. Models of the physical world may be given using differential equations that are obeyed at all times.

In this work, we analyze the periodically-scheduled controller subsystems of CPS using hybrid automata [2] and associated analysis tools. A hybrid automaton can directly model both the continuous behaviors and discrete aspects that arise when real-time scheduling and sampled control is combined with a continuously-evolving physical plant. The set of reachable states of a hybrid automaton, if it can be computed or overapproximated, can be used to formally prove control-theoretic properties about the system's transient and steady-state behavior. The controller subsystem models, after being proven correct, could then be integrated with hybrid automaton models of other parts of the system using modeling methods like hybrid input/output automata (HIOA) [3]. Reasoning about properties of the combined system could then be performed using assume-guarantee reasoning [4]. With such hybrid systems analysis, properties can be formally proven about *sets* of initial states as well as behaviors under bounded sensor error, actuator error, and other uncertainties. This has the potential to detect errors not found during simulation and testing, which deal with single initial states and specific execution traces.

Directly analyzing the controller subsystems of CPS using hybrid automaton reachability tools, unfortunately, does not usually work. One issue is that a large number of controller updates need to be considered in the analysis. The control code may need to be run tens or hundreds of times a second, and the physical system may need to evolve for tens of seconds to show the properties of interest. The number of discrete transitions that occur thus becomes extremely large. Real-time schedulers may also have variability in the exact scheduling time of the controller. Hybrid automaton reachability analysis tools perform poorly in such cases, with error bounds growing unacceptably large in the presence of many discrete transitions and timing uncertainty [5], [6].

In order to overcome these challenges, we apply a variant of the continuization technique [7], where a fast-switching hybrid system is abstracted by a continuous system with an additive nondeterministic input. We provide theoretical methods to compute bounds on the nondeterminism input needed for the continuization of periodically-scheduled controllers, which is essential for abstraction soundness. The developed approach is then automated using the Hyst [8] model transformation tool. In this way, we provide both a theoretical method that enables controller analysis with hybrid automaton reachability tools, and a practical way to use it.

The main contributions of this paper are:

- the modeling of periodically-controlled CPS using hybrid automata, with several models proposed based on possible implementation variations,
- the validated use of continuization to enable the analysis of these models, and a theoretical method to compute the bound on the nondeterminism globally as well as within local analysis domains,
- the implementation of the proposed technique in the Hyst model transformation tool, which allows rapid application to new hybrid automaton models, and
- a demonstration of the effectiveness of the proposed analysis approach on the design of a yaw damper system for a 747 jet aircraft.

DISTRIBUTION A. Approved for public release; Distribution unlimited. (Approval AFRL PA #88ABW-2015-2401, 15 MAY 2015)

In the next section, we present a brief background on modeling hybrid systems, give direct approaches for modeling real-time-scheduled controllers with hybrid automata, and provide reachability results showing scalability issues with these direct models. Next, Sec. III describes continuization and methods for computing the nondeterministic bounds it uses, which are essential for method accuracy. Then Sec. IV briefly describes the Hyst model transformation tool and the illustrates the continuization pass that implements the technique developed in this paper. Sec. V provides a case study showing the advantage of the analysis on a yaw damper control system for a 747 aircraft. A brief discussion of related techniques, especially a comparison versus classical control theoretic methods is given in Sec. VI, followed by a conclusion.

II. Hybrid Systems Modeling

The controller subsystem of a cyber-physical system (CPS) consists of a physical system interacting with a software controller, running periodically on a system using a real-time scheduler. A specific implementation can be formalized using a hybrid automaton model, and then its behavior, as well as the behavior of a composition of these subsystem models, can be analyzed using hybrid automata reachability tools. In this section, we elaborate on modeling controller subsystems using the hybrid automaton formalism. We first review hybrid automata (Sec. II-A), then propose three models that capture different possible implementations of a controller subsystem of a CPS (Sec. II-B). Finally, we attempt to directly perform reachability analysis of these systems using reachability analysis tools (Sec. II-C), which is shown to be challenging.

A. Preliminaries

A hybrid automaton is a formal model that captures both discrete behaviors as well as continuous dynamics present in a hybrid system. Roughly, it is a finite state machine with ordinary differential equations defined in each mode for a set of real-valued continuous variables.

Definition 1 (Hybrid Automaton)**.** *A* Hybrid Automaton *is a tuple*
$\mathcal{H}$ = (Loc, Var, Init, Flow, Trans, Inv) that defines:

- *a finite set of locations Loc,*
- *a set of n real-valued continuous variables Var = $\{x_1, \ldots, x_n\}$,*
- *an initial condition Init $\subseteq \mathbb{R}^n$ for each $\ell \in$ Loc,*
- *for each location ℓ, a relation Flow(ℓ) relating variables and their derivatives,*
- *a set of discrete transitions Trans, where each element is a tuple (ℓ, g, r, ℓ') with source location ℓ, guard g given as constraint on $\mathbb{R}^n$, reset r given as a function from $\mathbb{R}^n$to $\mathbb{R}^n$, and destination location ℓ',*
- *an invariant Inv(ℓ) $\subseteq \mathbb{R}^n$ for each location ℓ.*

A state of a hybrid system is a tuple $(\ell, \mathbf{X})$, where the discrete state is $\ell \in$ *Loc* and the continuous state $\mathbf{X}$ is a valuation—a mapping from a variable name to a point in the reals—of the continuous variables in *Var*.

Definition 2 (Trajectory)**.** *A* trajectory *of a hybrid system is an alternating sequence of continuous evolutions and discrete transitions, starting from a state in Init. Trajectories are subject to the following restrictions:*

- *the first state of the trajectory is an element of Init,*
- *during each continuous evolution, the continuous state evolves over an interval of real-valued time in accordance with the differential equations defined by Flow,*
- *during each continuous evolution, the continuous states always satisfy the location's invariant*[1]*, and*
- *during each discrete transition, the prestate is contained in transition's guard, and the change in state corresponds to applying the reset function to the continuous prestate and updating the location to ℓ'.*

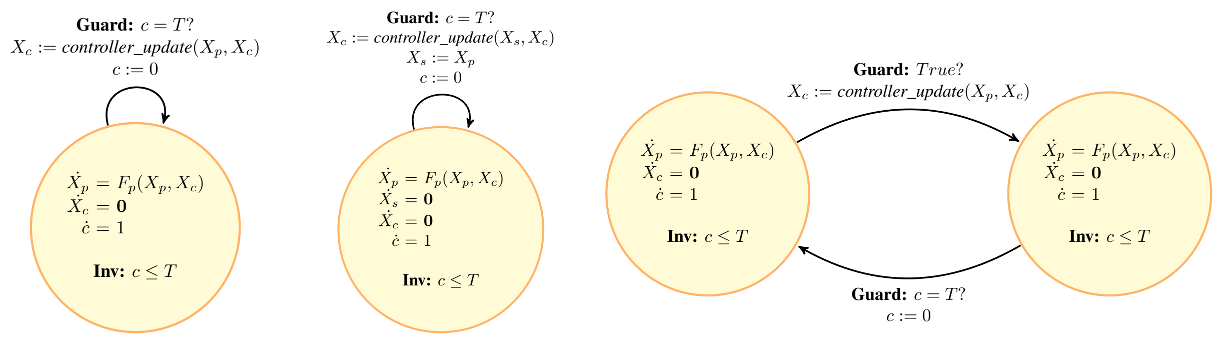
Definition 3 (Reachable Set)**.** *The set of all states that exist in any trajectory is called the* reachable set*. For a given hybrid automaton $\mathcal{H}$, we use* $\text{REACH}(\mathcal{H})$ *to denote the reachable set of $\mathcal{H}$. Given a subset of the variables $Y \subseteq$* Var *of hybrid automaton $\mathcal{H}$, the reach set projected onto those variables is written as* $\text{REACH}(\mathcal{H}) \downarrow Y$*. Typically we will be concerned with time-bounded reachable sets, where the amount of time that has elapsed during the continuous evolution portions of each trajectory is less than or equal to some given bound.*

B. CPS Modeling

We now describe three different ways that a CPS controller subsystem can be modeled using the hybrid automaton formalism, which correspond to different possible system implementations. First, we introduce the notion of a *Sampled CPS*, which has a continuous portion governed by differential equations, and a *controller_update* function that updates the discretely-controlled variables.

Definition 4 (Sampled CPS)**.** *A* Sampled CPS *is a system with n continuous variables divided into two groups. The first $np \leq n$ variables are the physical variables, and the remaining $nc = n - np$ variables are the cyber variables. The set of variables $Var = \{x_1, x_2, \ldots, x_n\}$ is partitioned into physical variables $X_p = \{p_1, p_2, \ldots, p_{np}\}$ and cyber variables $X_c = \{c_1, c_2, \ldots, c_{nc}\}$, where each variable $x_i \in \mathbb{R}$. Each physical variable has an associated differential equation, $\dot{p}_1 = f_1$, $\dot{p}_2 = f_2$, …, $\dot{p}_{np} = f_{np}$, where each $\dot{p}_i = f_i$ is a function $\mathbb{R}^n \rightarrow \mathbb{R}$. To ensure existence and uniqueness of the solutions, the differential equations are assumed to be Lipschitz continuous in the domain of interest. The dynamics for the physical variables are provided, so $F_p = (f_1, f_2, \ldots, f_{np})$ is given. The remaining nc variables are set periodically in control software, and remain constant between updates (zero-order hold). Their differential equations are given as $\dot{c}_1 = 0$, $\dot{c}_2 = 0$, …, $\dot{c}_{cn} = 0$. The control software is defined by a function controller_update : $\mathbb{R}^n \rightarrow \mathbb{R}^{nc}$, which updates the cyber variables based on the*

[1]If at some point the invariant were to become false, a discrete transition must be taken immediately. If no transition's guards are enabled, the model is said to deadlock as time cannot advance.



(a) Model 1 (b) Model 2 (c) Model 3

Fig. 1: Various hybrid automaton models formalize different implementations of a periodically-sampled CPS.

system state. The controller_update function can be decomposed into nc functions where each one updates a single cyber variable, $c_1 :=$ controller_update$_1(X_p, X_c), \ldots, c_{nc} :=$ controller_update$_{nc}(X_p, X_c)$. In this work, we will restrict the controller_update functions to ones that are differentiable and locally Lipschitz continuous in the input arguments, in the domain of interest (for example, discrete approximations of continuous controllers).

Model 1: The simplest model is for a strict periodic controller, where the control software runs with a given period, T. This could correspond to a system using a time-division multiple-access (TDMA) or other time-triggered scheduler, where the control task is nonpreemptive and the worst-case execution time (WCET) fairly short. In the model, a single location exists where time can elapse. An extra clock variable, c, is added to the hybrid automaton that ticks at rate one ($\dot{c} = 1$). When the clock reaches the period, a transition is forced by an invariant in the single location that $c \leq T$, which prevents continuous evolutions from continuing. The transition executes the controller logic when the clock reaches the period, then resets the clock to 0, and subsequently repeats periodically. A hybrid automaton visualization of this model is shown in Fig. 1(a). The strict periodic controller, however, does not exactly capture the behavior of a system using a real-time scheduler. A scheduler like rate-monotonic (RM) or earliest deadline first (EDF), provides a guarantee of execution *at some point within the period.*

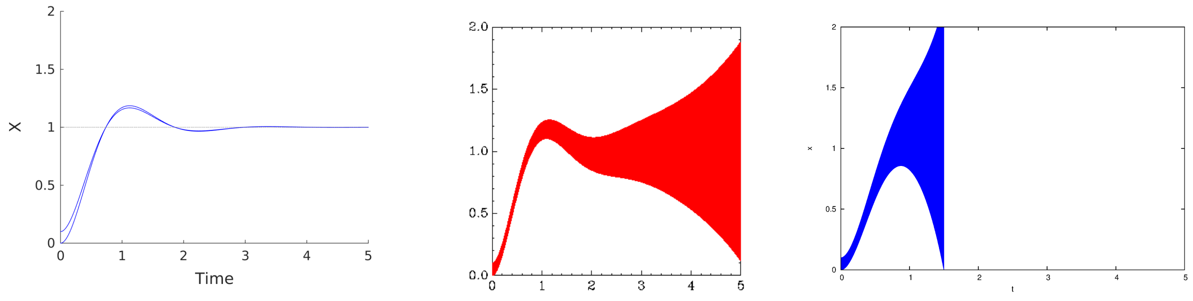
Model 2: An alternative implementation, which uses a real-time scheduler such as RM or EDF would sample the system at the start of the period, and write the actuation values at the end of the period. This can be modeled using a hybrid automaton by starting with the strictly periodic system (Model 1) and adding np additional cyber variables, which we call X_s, with derivatives equal to zero that model the sampled state. On the actuator assignment (*controller_update*) at the end of each period, the controller logic will then compute on the state sampled from the start of the period. After updating the cyber variables, the physical state would then be sampled again and stored into X_s for use at the end of the next period. The hybrid automaton model of this system is given in Fig. 1(b). The downside of such a controller is there is a one period delay introduced into the system, which may affect control performance, as well as np additional variables in the model, which may affect analysis scalability.

Model 3: An alternative implementation may consider directly sampling and actuating at some point during each period, where the sampling point is nondeterministic. This would be a reasonable model if the control task's execution is short and the task is non-preemptive. This model is similar to the strictly periodic Model 1 (Fig. 1(a)), except that: (1) a second mode is added to indicate if the controller has run yet during the period, (2) the first transition (the call to *controller_update*) happens nondeterministically up to the period T owing to the invariant $c \leq T$, and (3) the second transition (the end of the control period) happens when the clock reaches T time. The modified automaton is shown in Fig. 1(c). This model uses nondeterminism in discrete transitions to capture the type of guarantee provided by a real-time scheduler: that the control logic will execute and finish *at some point* within each period.

More complicated models could also be considered. For example, if the execution time was non-negligible or the task was preemptive, the state could be sampled nondeterministically at some point during the period minus the WCET, and then actuation could be performed nondeterministically up to the end of the period.

C. Preliminary Reachability Analysis

Although hybrid automata can model real-time scheduled controllers and plants as shown above, an important factor is tractability of analysis. Since analysis of even moderately-complicated hybrid automata is undecidable [9], tools often compute an overapproximation of the reachable states, which is sufficient for safety analysis (making sure unsafe states are not reachable). If the set of reachable states may be computed for unbounded time (if the reachability algorithm reaches a fixed-point) and the resulting set of states is bounded, then conclusions can also be drawn about system stability. In the presence of a large number of discrete switches, reachability analysis tools may significantly overapproximate the reachable set of states, due to the need to perform intersections of reachable sets with surfaces representing guard conditions [6]. These intersections are typically done geometrically, and result



(a) Simulations from $x \in \{0, 0.1\}$ (b) SpaceEx Reachability (c) Flow* Reachability

Fig. 2: The response for the periodically-controlled double-integrator system from Example 1 converges in simulation, but appears to diverge during reachability analysis.

in an overapproximation of the actual intersection, introducing some error at every discrete transition. Due to this concern, we empirically evaluate the performance of two modern reachability tools, SpaceEx [5] and Flow* [10], [11], on a simple control system using the approach from Model 1 (Fig. 1(a)).

Example 1 (Double-Integrator System)**.** *A double-integrator system, such as point moving along a 1-d line controlled through its acceleration, has two physical variables: x, its position, v, its velocity, and a single cyber variables a, its acceleration. The dynamics are $\dot{x} = v$, $\dot{v} = a$, and the acceleration a is set periodically by the control logic. There is a fixed setpoint the system tries to move towards at $x = 1$. The acceleration is set using a PD controller with gains $P = 10$ and $D = 3$. The controller_update function periodically assigns $a := P * (1 - x) + D * -v$. The period of the control task is $T = 0.005$ seconds (200 Hz). The initial states are $x \in [0, 0.1]$ and $v = 0$.*

Using the system in Example 1, we construct the corresponding hybrid automaton (shown in the Appendix in Fig. 8) and examine the controller's response. A control Lyapunov function may be derived to show stabilization of the purely continuous system to the setpoint of $x = 1$ and $v = 0$. In Matlab simulations of the periodically-sampled system from the boundary of the initial states (from both $x = 0$ and $x = 0.1$), the system easily converges to the setpoint. When performing reachability, however, both SpaceEx and Flow* produce divergent reachable sets, due to overapproximation error introduced at each of the discrete transitions. The simulations and reachability visualization are shown in Fig. 2.

Although effort was taken to optimize various tool parameters, they could likely be further adjusted to get a slightly better response. For this particular system, if the tools had built-in support for time-triggered transitions and could infer that the clock acts as a time-trigger for the discrete transition, the error in the computation could likely be reduced (although we could not find time-triggered support in either tool's documentation). However, this would not work for the nondeterministic switch in Model 3 (Fig. 1(c)), since that discrete transition (invocation of *controller_update*) can occur at any time within the period, based on the guarantees provided by schedulers like RM and EDF. The problem of accumulated error in reachability from many discrete transitions, in general, cannot be eliminated.

III. Continuization for Improved Analysis

The occurrence of many discrete transitions leads to accumulated error during reachability analysis because of a need to repeatedly take intersections of sets of states with the transition guards. One idea to get better accuracy, therefore, is to eliminate the discrete transitions altogether. Intuitively, this process relies on the observation that the behavior of the periodically-sampled system is contained in the behavior of the continuously-controlled system with some additional bounded nondeterministic input.

This process of validated abstraction of the sampled hybrid automaton by a continuous one is called *continuization* [12], and is briefly reviewed in the next subsection (Sec. III-A). Here, we apply the continuization idea in order to analyze periodic control systems, which has not been done before. This process relies on having a bound on the speed of changes of the cyber variables, and computing this bound is then described (Sec. III-B).

A. Continuization

Continuization is the process of abstracting a system with many discrete switches by a continuous one with an extra nondeterministic input. Previously, it was used to analyze rapidly-switching electric circuits [12], specifically locking time and stability properties for charge-pump phase-locked loops. The key challenge when performing continuization is determining the amount of nondeterministic input that is necessary in order to guarantee that all behaviors of the sampled system are captured by the continuous one, but not too much that analysis accuracy suffers.

In the earlier circuit work, this was done by solving for the change of state in one cycle with a known switching time. Since there was no closed-form solution for the switching time, interval analysis was performed using the ranges of possible switching times, and then this was used to derive conservative bounds on the change in state.

We want to apply continuization in order to analyze periodically-controlled CPS. We formalize this process by using sampling deviation functions.

Definition 5 (Sampling Deviation)**.** *A sampling deviation ω_i is a function $\mathbb{R} \rightarrow \mathbb{R} \times \mathbb{R}$, which, given a time, produces an upper and lower bound on the difference of a cyber variable $c_i \in X_c$, between its value in a sampled CPS and the update function controller_update(X_p, X_c).*

Given a sampling deviation function ω_i for each cyber variable, we can construct an overapproximation of the sampled CPS. First, we construct a continuous approximation of the sampled CPS.

Definition 6 (Continuous Approximation)**.** *A* continuous approximation *of a sampled CPS is a hybrid automaton where the controller logic is run continuously. That is, the discrete update for each cyber variables in $c_i \in X_c$ is removed from the system, and each cyber variable's differential equation is set to $\dot{c}_i = \frac{d}{dt}$controller_update$_i(X_p, X_c)$. The variable c_i's initial value is set to the value when the controller is run at the original initial state, controller_update$_i(X_p(0), X_c(0))$.*

The continuous approximation differs from the original sampled CPS. A *continuized abstraction* accounts for this difference by adding nondeterminism to every occurrence of each cyber variable within the continuous approximation.

Definition 7 (Continuized Abstraction, Continuization)**.** *A* continuized abstraction $\mathcal{H}_c$ *of a sampled CPS $\mathcal{H}$ is constructed starting from $\mathcal{H}$'s continuous approximation. Each occurrence of a cyber-variable c_i in the continuous approximation gets an extra term added equal to the sampling deviation ω_i. If any of the ω_i change over time, an additional time variable t is added to the system that starts at 0 and ticks at rate 1 forever.*

The model constructed using the above continuization approach will have trajectories of the physical variables that contain all the behaviors in the original sampled CPS.

Theorem 1 (Soundness of Continuization)**.** *Given a sampled CPS $\mathcal{H}$ as well as its continuized abstraction $\mathcal{H}_c$, $\text{REACH}(\mathcal{H}) \downarrow X_p \subseteq \text{REACH}(\mathcal{H}_c) \downarrow X_p$.*

Proof. Consider any cyber variable $c_i \in X_c$. Let $\text{Val}_{\text{sampled}}(c_i)$ be the value of the variable in the sampled CPS, and $\text{Val}_{\text{abstract}}(c_i)$ be the value of the variable in the continuized abstraction. At any time t in a trajectory, we first show that $\text{Val}_{\text{sampled}}(c_i) \in \text{Val}_{\text{abstract}}(c_i) + \omega_i(t)$.

By the definition of the sampling deviation function, the difference between $\text{Val}_{\text{sampled}}(c_i)$ and *controller_update*$_i(X_p, X_c)$ at time t must be contained in the interval $\omega_i(t)$. Therefore, $\text{Val}_{\text{sampled}}(c_i)$ is contained in *controller_update*$_i(X_p, X_c) + \omega_i(t)$. The continuous approximation at time t is equal to *controller_update*$_i(X_p, X_c)$, and by the construction of the continuized abstraction from the continuous approximation, the inclusion $\text{Val}_{\text{sampled}}(c_i) \in \text{Val}_{\text{abstract}}(c_i) + \omega_i(t)$ holds.

In the construction of the continuous abstraction, each cyber variable c_i in the continuous approximation was replaced by $c_i + \omega_i(t)$. Since, as shown above, $\text{Val}_{\text{sampled}}(c_i) \in \text{Val}_{\text{abstract}}(c_i) + \omega_i(t)$, the derivatives for every variable in the sampled CPS will be contained in the derivatives of continuized abstraction. In particular, the physical variable values in the continuized abstraction also contain the sampled CPS physical variable values. The discrete transitions between the two systems are identical, except for the removal of the periodic cyber-variable updates in the continuized abstraction. Thus, any discrete transition (other than controller updates, which only update cyber variables and for which we already showed containment) taken by the sampled CPS can also be taken by the continuized version. Since a trajectory is an alternating sequence of continuous evolutions and discrete transitions, and the initial states are the same, by induction on the length of a trajectory, the values of the physical variables in the sampled CPS are always contained in the values of the physical variables in the continuized abstraction. Therefore, $\text{REACH}(\mathcal{H}) \downarrow X_p \subseteq \text{REACH}(\mathcal{H}_c) \downarrow X_p$. □

B. Producing Sampling Deviation Functions

The key to continuization is to construct sampling deviation functions that provide an upper and lower bound on the difference of each cyber variable between the sampled CPS and the *controller_update* function. One way to compute such a function is by looking at the maximum rate of change (bounded by a Lipschitz constant) of the derivative of each cyber variable in the continuous approximation. This process makes use of standard interval arithmetic multiplication, $[a, b] * [c, d] = [\min(a*c, a*d, b*c, b*d), \max(a*c, a*d, b*c, b*d)]$.

Lemma 1 (Sampling Deviation using Lipschitz Constant)**.** *Given interval bounds, $K = [K^{min}, K^{max}]$, on the rate of change of the derivative of c_i in the continuous approximation, and the period of the associated strictly-periodic (Model 1 from Fig. 1) controller, T, a sampling deviation function is $\omega_i = [-T, 0] * K$.*

Proof. The sampling deviation function needs to bound the difference of the value of the variable c_i in a sampled CPS and *controller_update*$_i(X)$. The difference between *controller_update*$_i$ at the last sample time (which is the current value of the cyber variable in the sampled CPS), and *controller_update*$_i$ at the current time (which is its value in the continuous approximation) is at most a product of the maximum rate of change K, and the time since the last sample. The difference between the last controller update and the current time must be in the interval $[-T, 0]$, since it is a strictly periodic controller with period T. Assuming the first sample occurs at time 0, by induction on the number of samples, this property will hold for every sampling period and therefore over all time. □

In this case, we had considered a strictly periodic controller, such as the one given by Model 1 in Sec. II-B. To compute the function for a nondeterministic controller such as Model 3, all that would need to be adjusted is the time of the last controller update. In the worst case, a sample will occur at the start of one period, and at the end of the next period. In that case, the maximum time between updates is $2 * T$, so using an interval of $[-2 * T, 0]$ in Lemma 1 would be adequate.

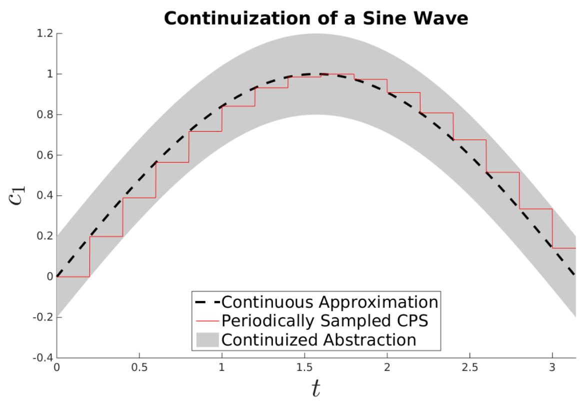


Fig. 3: The main idea behind the proposed continuization approach is that a nondeterministic continuous system contains the behaviors of a periodically sampled system.

To provide some intuition on the construction of sampling deviation functions, we provide an simple illustrative example.

Example 2 (Sine Wave)**.** *Consider a system with a single cyber variable c_1 where the controller_update function is given by $\sin(t)$, and t is a clock (physical variable with $\dot{t} = 1$) ticking from 0 to π. The period of the cyber-variable is $T = 0.2$.*

The rate of change of *controller_update* (the derivative) is equal to $\cos(t)$, and the bound on cos in $[0, \pi]$ is $K = [-1, 1]$. Given this bound and the period of $T = 0.2$, each occurrence of c_i in the continuous approximation is replaced by $c_i + [-0.2, 0.2]$ in the continuized abstraction. A visual depiction of this is given in Fig. 3.

One nice property of the sampling deviation function constructed by Lemma 1 is that no matter how large the bounds are on the rate of change of the *controller_update* function, the sampling deviation function can be made arbitrarily small by choosing a small enough controller period T. This is because of the multiplication in the sampling deviation function by the interval $[-T, 0]$. Intuitively, this makes sense, since the continuous system is more closely approximated as we sample and actuate at a higher frequency. This is in contrast, however, to reachability analysis done directly on the sampled CPS models, where smaller periods lead to more discrete transitions, which lead to more error.

The width of the interval given by deviation function does affect the amount of overapproximation in the constructed model, and therefore it is desirable to have this function be as tight as possible. One way to improve the bound on the rate of change of the cyber variable is by considering smaller domains (time intervals). For example, we could take advantage of the time dependence of each sampling deviation function ω_i, and define corresponding sampling deviation functions within local analysis domains.

Lemma 2 (Sampling Deviation in Local Analysis Domains)**.** *For a cyber variable c_i with period T, given a sequence of interval bounds on the rate of change of the controller_update function, K_1, K_2, …, K_m, and an associated sequence of increasing and pointwise-intersecting time intervals (which we call local analysis domains) where the bounds are valid, $[t_0 = 0, t_1], [t_1, t_2], \ldots, [t_{m-1}, t_m]$, a sampling deviation function up to time t_m can be computed as:*

$$\omega_i(t) = [-T, 0] * [\min(\{K_j^{min} \mid t \in [t_{j-1}, t_j + T]\}), \max(\{K_j^{max} \mid t \in [t_{j-1}, t_j + T]\})].$$

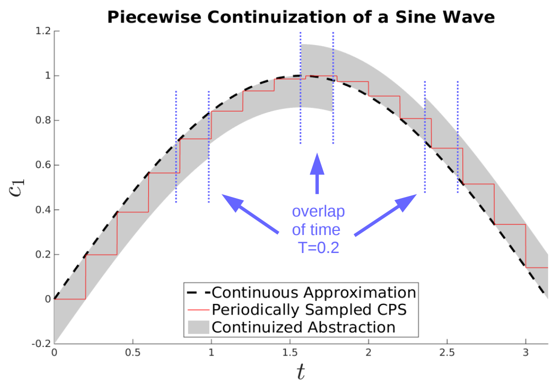


Fig. 4: The continuization approach as applied to four local analysis domains has an overlap of one period length between domains.

Proof. Notice the time intervals have the controller period T added to the upper time bound. This is because when a new time interval is entered in a trajectory, the sampled CPS could have taken the most-recent sample in either the current time interval, or in the previous one. The sampling deviation function, therefore, must account for both possibilities until T time has elapsed in the new interval. Other than this caveat, the proof follows that of Lemma 1, except that the analysis is done at each time interval. □

In the sine wave system from Example 2, we can apply this approach in four analysis domains (time intervals), $[0, \frac{\pi}{4}], [\frac{\pi}{4}, \frac{\pi}{2}], [\frac{\pi}{2}, \frac{3\pi}{4}], [\frac{3\pi}{4}, 1]$. Solving for the $\cos(t)$ (the derivative of $\sin(t)$) in these domains, we can come up with the associated interval bounds on $\dot{c}_1$ in the continuous approximation, $K_1 = [\frac{\sqrt{2}}{2}, 1], K_2 = [0, \frac{\sqrt{2}}{2}], K_3 = [-\frac{\sqrt{2}}{2}, 0], K_4 = [-1, -\frac{\sqrt{2}}{2}]$. Using the period $T = 0.2$, we then obtain the piecewise continuization of the system, shown in Fig. 4.

In Fig. 4, when the derivative bounds are positive, the difference between the sampled CPS and the continuous approximation is negative, which is why an interval of $[-T, 0]$ was used to bound the difference. Also, without the presence of the overlap between time domains, the continuized abstraction would be wrong immediately after time $\frac{\pi}{2}$. In this case, the new domain has a strictly negative derivative, but because the sample occurred before $\frac{\pi}{2}$, the bound from the previous domain must be used.

There are two considerations when applying continuization with local analysis domains. First, the result is only valid until the maximum time of the last analysis domain. If this time is finite, this means only bounded-time reachability can be computed. Second, there is a trade off between the accuracy of the computation and the number of domains considered. Continuization was originally used to eliminate large numbers of discrete transitions in a sampled CPS. Using local analysis domains, however, brings back discrete transitions, although now the number of transitions can be controlled by adjusting the number of domains. Using too many domains may lead to similar problems with tool performance as when we directly considered a sampled CPS model for reachability analysis. We could solve this problem by having sampling deviation functions that vary as continuous functions of time, although the way to create these is less clear, and left as possible future work.

IV. Automation in Hyst

Hyst [8] is a model transformation and translation tool for hybrid automaton models. Hyst performs both model *translation*, which converts between formats of different reachability tools, as well as model *transformations*, which serve to improve reachability computation results. The continuization approach described in the previous section has been implemented as a model transformation pass in Hyst, which permits easy application of the developed technique.

A. Transformation Pass

The implemented model transformation pass performs continuization starting given a continuous approximation of the system. The user provides (1) a target model file describing the hybrid automaton, (2) the controller period, T, (3) the name of the cyber variable of interest, c_i, (4) a sequence of m increasing times used to construct local analysis domains, (5) a corresponding sequence of m bloating terms, which will be described shortly, and (6) the name of the time variable (optional; only used if multiple local analysis domains are used to create transitions between them).

Given these inputs, the pass first simulates the continuous abstraction from the center of the initial states, in order to approximate the interval bounds on the rate of change of the derivative of c_i. For each time interval, the bound during that time is then expanded by the corresponding user-provided bloating term. We call the new intervals *candidate Lipschitz bounds* for the cyber variable's derivative. The candidate Lipschitz bounds are used as described in Lemma 2, along with the time domains, in order to produce the sampling deviation function $\omega_i(t)$.

The sampling deviation function consists of piecewise constant intervals. For each piece, a mode is created in the output hybrid automaton, with dynamics equal to the continuous approximation, except with every occurrence of c_i replaced by $c_i + \omega_i(t)$. Transitions are then added between the modes when the appropriate amount of time has elapsed.

The bound given by ω_i is only valid, however, if the candidate Lipschitz bounds are actually upper and lower bounds on the derivative of the cyber variable. This can happen because the bounds are constructed from a single simulation using the continuous approximation, whereas the reachable set of states considers all initial points as well as the expanded set of values for the cyber-variable in the dynamics, $c_i + \omega_i(t)$ instead of just c_i. To check if the bounds are respected, invariants and guards are added to the output hybrid automaton to check if the derivative exceeds the candidate Lipschitz bounds. If a violation occurs, a transition to an error state is taken, which is added as a forbidden location in the model. *In this way, performing reachability computation will not only give the set of states reachable by the continuized abstraction, but will also check that the candidate Lipschitz bounds are actual bounds on the derivative of the cyber variable.* If they are not, the transition to the error state will be detected when performing a reachability computation, and the transformation pass can be re-run with larger bloating terms, which will increase the size of the candidate Lipschitz bounds.

B. Example

We apply the continuization approach in Hyst to the double-integrator system given in Example 1. The *controller_update* function in this case is $P * (1 - x) + D * -v$, with $P = 10$ and $D = 3$. The time derivative is $-10 * \dot{x} - 3 * \dot{v}$. After substituting in the derivatives ($\dot{x} = v$, $\dot{v} = a$), the derivative of a in the continuous abstraction is: $-10 * v - 3 * a$. The initial value of a is the value assigned when *controller_update* is evaluated at the initial states, $a := 10 * (1 - x) + 3 * -v$. The hybrid automaton of the continuous approximation shown in the appendix, in Fig. 9.

The pass implemented in Hyst performs a simulation of the system starting from the center of the initial set of states, in this case, at $x = 0.05$, $v = 0$, $a = 9.5$. The value of $\dot{a}$ in the simulation is observed to be in the interval $[-28.64, 5.27]$. This interval is then bloated by the provided bloating term, for which we consider ± 1, ± 2 and ± 4.

When running reachability with a bloating term of 1, Flow* immediately (at time 0) detects that the constructed error states are reachable, which means that the candidate Lipschitz bounds do not contain all the encountered values of $\dot{a}$. Computationally, we can show this to be the case. Initially, $x = [0, 0.1]$, which means the initial value of a is $[9, 10]$. The initial value of $\dot{a}$ is $-10 * v - 3 * a = [-30, -27]$. The interval values of $\dot{a}$ in the simulation were $[-28.64, 5.27]$, which bloated by 1 give candidate Lipschitz bounds of $[-29.64, 6.27]$. The lower bound of the derivative of the cyber variable (-30) is initially outside of the candidate bounds, which was detected by the transition to the error state.

Using a bloating term of 2, the candidate Lipschitz bounds are $[-30.64, 7.27]$, which contain the above-computed initial values of $\dot{a}$. When performing reachability, however, at time 0.04 an error state is reached again. At this time, the reachable set contains a state where $a = 8.79$ and $v = 0.382$. In this case, the derivative $\dot{a} = -10 * v - 3 * a + \omega_i = -10 * 0.382 - 3 * 8.79 + [-0.45, 0.11]$ has a lower value of -30.66, which is below the candidate Lipschitz bound of -30.64.

When the larger bloating term of 4 is used, the candidate Lipschitz bound is respected by the reachable set, and Flow* does not reach the out-of-bounds error states. Thus, the reach

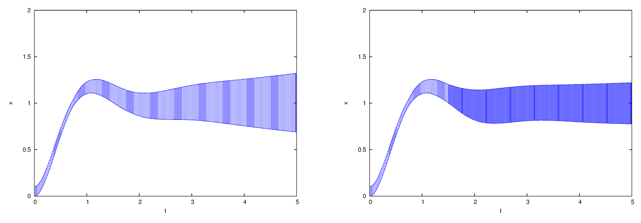

(a) Continuized System (b) Two Analysis Domains

Fig. 5: The response for the continuized periodically-controlled double-integrator system from Example 1 is significantly tighter than direct analysis (Fig. 2).

set of the continuized abstraction is a validated overapproximation of the reach set of the sampled CPS.

Recall, however, that directly computing the reach set of the sampled CPS, as shown in Fig. 2, resulted in a large exponential blow up in the size of the reachable set due to accumulation of overapproximation error. Even with a single analysis domain, the reachable set is significantly smaller, as shown in Fig. 5(a). Using multiple analysis domains, the reachable set can be further reduced. The hybrid automaton of the continuized system with two local analysis domains $[0, 1.5]$ and $[1.5, 5]$ is shown in the appendix in Fig. 10. The reach set of the response for this system is shown in Fig. 5(b). *Thus, the continuization method developed in this paper enables a more precise formal analysis of this system using hybrid automaton reachability tools.*

In terms of overhead, the runtime of the pass itself is small, taking about 100 ms. The reachability computation takes 0.9 seconds for the single-domain case, and about 1.3 seconds for the two-domain system, which is significantly faster than the 12 minutes needed for SpaceEx to produce Fig. 2(b).

V. Case Study

In this section, we apply the technique developed from Sec. III in order to perform reachability analysis of a hybrid system model of a yaw-damper for a 747 aircraft.

A. System and Controller Model

The model and controller we analyze in this case study are taken from the Control Systems Toolbox case studies in Matlab [13]. In brief, the system is a multiple-input multiple-output (MIMO) system that uses the aileron and rudder in order to reduce oscillations in the yaw and roll angle.

The analysis of the yaw damper is done on the system's aileron-to-bank angle impulse response. Three different systems are considered: (1) the original, undamped system, which experiences oscillations upon an impulse input, (2) the system with proportional compensator, which eliminates the oscillations but also over-stabilizes the spiral mode (a desired characteristic for the control), and (3) the system with a washout filter, which eliminates the oscillations but keeps the spiral mode.

We use this case study to evaluate the developed continuization technique so as to evaluate properties about the response of the final (washout filter) system. There are four

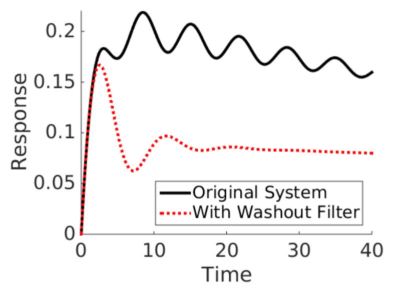


Fig. 6: The impulse response for the washout filter design of a yaw damper demonstrates the spiral mode in simulation.

physical variables in this system, sideslip angle (x_1), yaw rate (x_2), roll rate (x_3), and bank angle (x_4), represented by the column vector x. The two inputs u, are the rudder (u_1) and aileron (u_2). The outputs are the yaw rate and bank angle. The dynamics for the physical system are the standard linear time-invariant dynamics, $\dot{x} = Ax + Bu$ (the A and B matrices are provided in the in Sec. B of the appendix).

This physical system is put into a feedback loop with a washout filter. The washout filter has a single variable, w, with dynamics $\dot{w} = x_2 - 0.2 * w$. The washout filter variable is combined with the yaw to produce an effect on the rudder input. That is, the washout filter adds to u_1 the value $2.34 * (x_2 - 0.2 * w)$.

A simulation of the aileron-to-bank angle impulse response from this system, with and without the washout filter, is given in Fig. 6. In particular, the two control properties of interest are a lack of oscillations (quick settling time), and the presence of the spiral mode. The spiral mode is a desirable flight characteristic demonstrated by the apparent[2] steady-state offset in the rudder-to-bank angle impulse response.

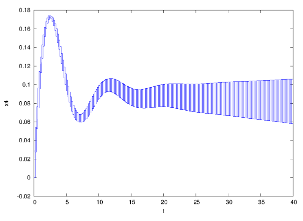
A property to check is that the aileron to bank angle impulse response remains around the simulated value of 0.08, between 20 and 40 seconds, and thus maintains the spiral mode without significant oscillation. We consider a controller running at 20 Hz ($T = 0.05$), using the implementation that samples and actuates when the real-time scheduled controller runs (Model 3 from Sec. II-B).

B. Reachability Analysis

Neither SpaceEx nor Flow* can effectively compute reachability on the periodically-actuated system model (Fig. 11 in the appendix). The reachable set of states explodes almost immediately, and neither tool can compute accurate time-bounded reachability for the required 40 seconds.

We apply the continuization approach developed in this paper by using the Hyst transformation pass on the continuous approximation of the model. First, we apply the technique over the whole time range. Initially, we try a small bloating term, and increase it until error states are no longer reachable during analysis. For the period parameter given to the pass, we use twice the control period, as this is needed to account for the

[2]The steady state is actually zero, but the convergence is very slow over hundreds of seconds.


(a) Reachability in Flow* of the Continuized Model

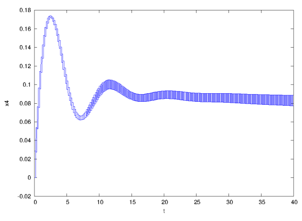

(b) Reachability with Local Domains and Halving Period

Fig. 7: Flow* can successfully compute reachability on the continuized model. When a smaller period and local analysis domain is used, the result is tighter.

maximum delay in sampling in Model 3, as discussed earlier in Sec. II-B. Flow* successfully computes reachability for the model, and confirms that the final bloating term (0.0007) was sufficiently large. The output plot is shown in Fig. 7(a).

Although the computation completes, which is an improvement over the direct computation, the set of states appears to be diverging slowly. The reachability result can be improved by using local analysis domains, or by reducing the controller period. To demonstrate this, we halve the controller period, and use two analysis domains. For time $[0, 8]$ we use a bloating term of 0.0004, and for time $[8, 40]$ we use 0.0003. Hyst creates the associated model file for Flow*, which we then use to compute reachability. Flow*, in about 5 seconds, confirms that the candidate domains are sufficient, and the resultant reachability plot is tighter than the previous one, as shown in Fig. 7(b). Furthermore, the spiral mode can be observed from the reachable set plot, along with the absence of oscillations in the time range $[20, 40]$.

VI. Related Work

In this paper, we have focused on controller analysis using hybrid automata reachability tools, although there are existing methods in control theory to design and analyze controllers. The design of a controller for a continuous-time system often occurs in continuous-time, and the controller is subsequently discretized[3] to be implemented in a software controller that operates periodically.

Continuous-Time Controller Design: There are many methods for control design in continuous-time. For example, a common strategy for linear time-invariant (LTI) systems is to design a stabilizing linear state-feedback controller of the form $u = Kx$ for a vector K [16]. Assuming the system is both controllable and observable, the strategy yields a new closed-loop system: $\dot{x} = Ax + Bu$ for $u = Kx$. After substituting this gives $\dot{x} = Ax + B(Kx)$ and then $\dot{x} = (A + BK)x$. This strategy is also known as pole placement [16]. Finding the vector K such that $(A + BK)$ is exponentially stable can be formulated in a variety of ways, such as by solving a linear matrix inequality (LMI) [17]. Linear quadratic regulator (LQR) design is another linear system design technique that also incorporates a cost function to yield an optimal controller [18]. LQR is used within the Linear Quadratic Gaussian (LQG) problem that robustly tolerates Gaussian additive noise inputs from disturbances. Other control design methods for linear systems are performed in the frequency domain, where pole and zero placement may also be performed to ensure stability and analyze performance criteria such as gain margins, phase margins, and use graphical tools like Nyquist diagrams and Bode plots. Design of controllers for nonlinear systems is challenging, but many approaches exist, such as linearizing and using gain-scheduled linear controllers, backstepping, feedback linearization, and many others [19].

[3]In this paper, we only focus on the conversion from continuous-time to discrete-time, and do not consider full digitization [14], [15], for example, the conversion from continuous-time and continuous-state to discrete-time and discrete-state through quantization.

Discretization of Continuous Controllers: Discretization typically consists of several steps. First, a sampling period must be selected at which measurements of the physical system are taken and made available to the software controller. Second, a control period must be selected to specify the rate at which control decisions are produced by the software controller and sent to actuators to influence the plant. Typically, these periods are selected in accordance with the speeds of the dynamics, and a common rule of thumb is to use the Nyquist frequency of the physical process to determine the minimum sampling period. The Nyquist frequency is twice the highest waveform frequency.

Given these periods, a discrete-time version of the plant can be constructed (using the sampling period) and a discrete-time version of the controller can be constructed (using the control period). Both discretizations are needed, as from the perspective of the controller, it will only receive state measurements of the plant at the points in time specified by the sampling period.

Discrete Controllers with Continuous Plants: While from the perspective of the software controller, the changes to the plant occur discretely, in reality, the plant evolves continuously according to differential equations. Controller performance with such constraints has been extensively investigated, and tools like JitterBug and TrueTime can characterize controller performance with real-time constraints and delays [20]. More recent works aid in synthesizing embedded software from hybrid systems models [21]. Giotto aids in this process of moving from control models to embedded real-time code [22].

Reachability: The elimination of large numbers of discrete transitions in hybrid automata was previously accomplished by continuization [7]. The earlier work was used to analyze properties about fast-switching electronic circuits. This work, in contrast, applied continuization to enable the analysis of fast-switching hybrid automata resulting from the periodic interactions with the real-time scheduler. We also considered using local analysis domains to construct the nondeterministic term, which was shown to increase the accuracy of the model.

Periodically Controller Hybrid Automata (PCHA) is one formalism for periodically-controlled embedded systems [23]. Automated analysis of PCHAs is possible only if the vector fields are polynomial, whereas, using the developed Hyst pass, continuization can be automatically applied to a broader class of systems. Combinations of reachability tools and SMT solvers have been used to model both physical-world dynamics and software behavior [24]. A limitation of this approach is that cyber-variables are represented with intervals, and that only strictly-periodic systems can be analyzed (Model 1 from Sec. II-B).

VII. Conclusion

Analysis of large CPS using formal hybrid systems analysis techniques remains difficult. A challenge problem was recently proposed to the research community by Toyota on the verification of a powertrain control system [25]. Although initial progress has been made on simplified versions of the system [26], the full benchmark model presents four main challenges for verification tools: (1) controllers that periodically actuate the plant, (2) lookup tables to describe the system dynamics, (3) the presence of time delays in the model, and (4) large system scale.

In this paper, we addressed the first of these issues, by using continuization in order to soundly abstract the periodically-controlled dynamics. This permits initial analysis of these systems using reachability tools for hybrid automata. Without our approach, existing tools produce exponentially divergent reach sets on these models, and often fail before reaching the desired time bound. Since the accuracy of analysis depends on the tightness of the difference between the discrete system and continuized abstraction, a possible future improvement would be to compute these bounds in local domains based on the system state, in addition to time as proposed in this paper.

Acknowledgments

This material is based on research sponsored by the Air Force Research Laboratory under agreement number FA8750-15-1-0105, the Air Force Office of Scientific Research (AFOSR), in part through the Summer Faculty Fellowship Program (SFFP) and contract FA9550-15-1-0258. This material is based upon work supported by the National Science Foundation (NSF) under Grant Nos. 1464311 and 1527398. 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 AFOSR, AFRL, or NSF.

References

[1] 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.

[2] 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.

[3] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid i/o automata," *Inf. Comput.*, vol. 185, no. 1, pp. 105–157, Aug. 2003.

[4] S. Bogomolov, G. Frehse, M. Greitschus, R. Grosu, C. S. Pasareanu, A. Podelski, and T. Strump, "Assume-guarantee abstraction refinement meets hybrid systems," in *Hardware and Software: Verification and Testing - 10th International Haifa Verification Conference, HVC 2014, Haifa, Israel, November 18-20, 2014. Proceedings*, 2014, pp. 116–131.

[5] 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.

[6] M. Althoff and B. H. Krogh, "Avoiding geometric intersection operations in reachability analysis of hybrid systems," in *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '12. New York, NY, USA: ACM, 2012, pp. 45–54.

[7] M. Althoff, S. Yaldiz, A. Rajhans, X. Li, B. Krogh, and L. Pileggi, "Formal verification of phase-locked loops using reachability analysis and continuization," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, Nov 2011, pp. 659–666.

[8] S. Bak, S. Bogomolov, and T. T. Johnson, "HyST: A source transformation and translation tool for hybrid automaton models," in *18th International Conference on Hybrid Systems: Computation and Control (HSCC 2015)*. Seattle, Washington: ACM, Apr. 2015.

[9] 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.

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

[11] ——, "Flow*: An analyzer for non-linear hybrid systems," in *International Conference on Computer Aided Verification (CAV)*, 2013.

[12] M. Althoff, A. Rajhans, B. H. Krogh, S. Yaldiz, X. Li, and L. Pileggi, "Formal verification of phase-locked loops using reachability analysis and continuization," *Commun. ACM*, vol. 56, no. 10, pp. 97–104, Oct. 2013.

[13] T. Mathworks, "Yaw damper design for a 747 jet aircraft," Control System Toolbox Examples, 2015. [Online]. Available: http://http://www.mathworks.com/help/control/examples/yaw-damper-design-for-a-747-jet-aircraft.html

[14] R. Brockett and D. Liberzon, "Quantized feedback stabilization of linear systems," *Automatic Control, IEEE Transactions on*, vol. 45, no. 7, pp. 1279–1289, Jul. 2000.

[15] T. T. Johnson, S. Mitra, and C. Langbort, "Stability of digitally interconnected linear systems," in *Proceedings of the 50th IEEE Conference on Decision and Control and European Control Conference (CDC ECC 2011)*, Orlando, Florida, USA, Dec. 2011, pp. 2687–2692.

[16] C. Chen, *Linear System Theory and Design*, 3rd ed. New York, NY: Oxford University Press, 1999.

[17] J. Löfberg, "Yalmip : A toolbox for modeling and optimization in MATLAB," in *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004. [Online]. Available: http://users.isy.liu.se/johanl/yalmip

[18] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal control*. John Wiley & Sons, 2012.

[19] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[20] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. Arzen, "How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime," *Control Systems, IEEE*, vol. 23, no. 3, pp. 16–30, June 2003.

[21] M. Anand, S. Fischmeister, Y. Hur, J. Kim, and I. Lee, "Generating reliable code from hybrid-systems models," *Computers, IEEE Transactions on*, vol. 59, no. 9, pp. 1281–1294, Sep. 2010.

[22] T. Henzinger, C. Kirsch, M. Sanvido, and W. Pree, "From control models to real-time code using giotto," *Control Systems, IEEE*, vol. 23, no. 1, pp. 50–64, 2003.

[23] T. Wongpiromsarn, S. Mitra, A. Lamperski, and R. M. Murray, "Verification of periodically controlled hybrid systems: Application to an autonomous vehicle," *ACM Trans. Embed. Comput. Syst.*, vol. 11, no. S2, pp. 53:1–53:24, Aug. 2012.

[24] G. Simko and E. K. Jackson, "A bounded model checking tool for periodic sample-hold systems," in *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '14. New York, NY, USA: ACM, 2014, pp. 157–162.

[25] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, "Powertrain control verification benchmark," in *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '14. New York, NY, USA: ACM, 2014, pp. 253–262.

[26] C. Fan, P. S. Duggirala, S. Mitra, and M. Viswanathan, "Progress on powertrain verification challenge with C2E2," in *ARCH '15: Proc. of the 2nd Workshop on Applied Verification for Continuous and Hybrid Systems*, April 2015.

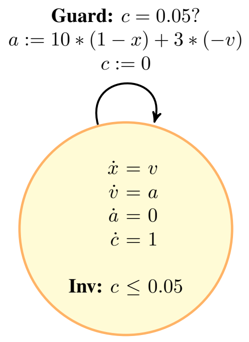


Fig. 8: Hybrid automaton model for sampled CPS of the double-integrator system in Example 1.

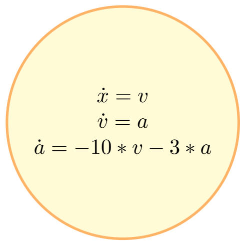


Fig. 9: Hybrid automaton model for continuous approximation of the double-integrator system in Example 1.

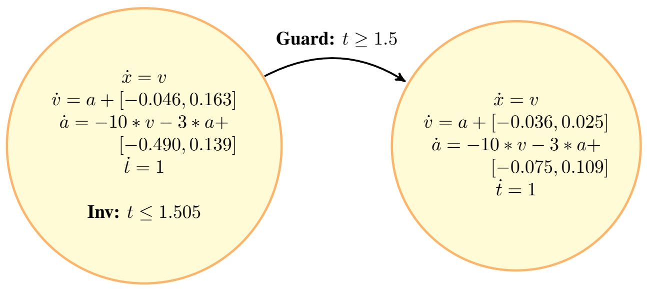


Fig. 10: Hybrid automaton model for continuized abstraction with two analysis domains (with error modes and transitions omitted) of the double-integrator system in Example 1.

Appendix

A. Double-Integrator Example

The hybrid automata for the double-integrator system (Example 1) are shown in Figs. 8, 9, and 10. In the continuous approximation and the continuized abstraction, the initial value of a is taken to be the value when *controller_update* is evaluated at the initial states, $a := 10 * (1 - x) + 3 * -v$.

The continuized abstraction shown in Fig. 10 is constructed from two time domains, $[0, 1.5]$ and $[1.5, 5]$, using a bloating term of 4 for each of the domains. The ranges of $\dot{a}$ in simulation for the two domains are $[-28.65, 5.27]$ and $[-0.97, 3.24]$, which give interval bounds of $K_1 = [-32.65, 9.27]$, and $K_2 = [-4.97, 7.24]$. With a period of $T = 0.005$, this gives interval values for ω of $[-0.046, 0.163]$ and $[-0.036, 0.025]$. The derivative $\dot{a}$ uses a value of -3 multiplied by these intervals due to the substitution of a by $a + \omega$ (since a is multiplied by -3 in the derivative). The derivative could have equivalently been written as $\dot{a} = -10 * v - 3 * (a + [-0.036, 0.025])$.

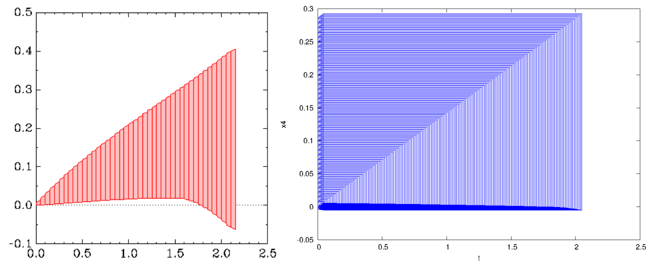


Fig. 11: Neither SpaceEx (left) nor Flow* (right) can directly compute reachability accurately on the yaw-damper model.

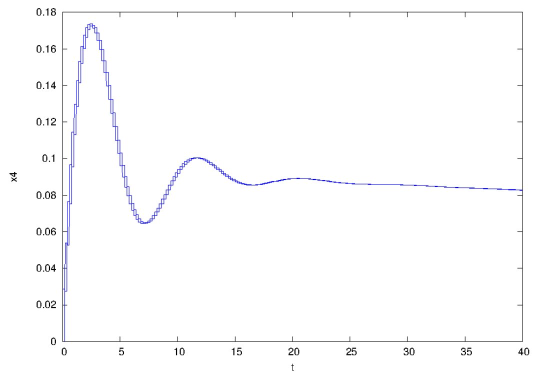


Fig. 12: The continuous approximation of the yaw-damper system demonstrates the spiral mode.

In Fig. 10, the error modes and transitions were not drawn. The guard conditions to enter an error mode in the first domain are $-10*v-3*a+0.139 \geq 9.27$ or $-10*v-3*a+-0.490 \leq -32.65$. In the second domain, the guard conditions are $10 * v - 3 * a + 0.109 \geq 7.24$ or $-10 * v - 3 * a + -0.075 \leq -4.97$.

B. Yaw-Damper Example

The dynamics of the yaw-damper system from Sec. V are standard linear time-invariant dynamics, $\dot{x} = Ax + Bu$, with:

$$A = \begin{bmatrix} -0.0558 & -.9968 & 0.0802 & 0.0415 \\ 0.598 & -0.115 & -0.0318 & 0 \\ -3.05 & 0.388 & -0.4650 & 0 \\ 0 & 0.0805 & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.00729 & 0 \\ -0.475 & 0.00775 \\ 0.153 & 0.143 \\ 0 & 0 \end{bmatrix}.$$

Neither SpaceEx nor Flow* can compute reachability on the periodically-actuated system. The reachability plots produced by the reachability tools on the real-time actuated model (Model 3) are given in Fig. 11.

The continuous approximation of the system demonstrates the spiral mode and is close to the reach set for the periodically-actuated washout filter system. The plot for the continuous approximation is shown in Fig. 12. This is the system that is used as input to the Hyst continuization pass.