SpaceEx:
Scalable Verification of Hybrid Systems

Goran Frehse[1], Colas Le Guernic[2], Alexandre Donzé[1], Scott Cotton[1],
Rajarshi Ray[1], Olivier Lebeltel[1], Rodolfo Ripado[1], Antoine Girard[3],
Thao Dang[1], Oded Maler[1]

[1] Verimag, CNRS / Université Grenoble 1 Joseph Fourier, 38610 Gières, France
[2] New York University CIMS, New York, NY 10012, USA
[3] Laboratoire Jean Kuntzmann, Université de Grenoble
goran.frehse@imag.fr

Abstract. We present a scalable reachability algorithm for hybrid systems with piecewise affine, non-deterministic dynamics. It combines polyhedra and support function representations of continuous sets to compute an overapproximation of the reachable states. The algorithm improves over previous work by using variable time steps to guarantee a given local error bound. In addition, we propose an improved approximation model, which drastically improves the accuracy of the algorithm. The algorithm is implemented as part of SpaceEx, a new verification platform for hybrid systems, available at `spaceex.imag.fr`. Experimental results of full fixed-point computations with hybrid systems with more than 100 variables illustrate the scalability of the approach.

1 Introduction

Hybrid systems are a class of mathematical models of dynamical systems admitting both discrete-event (logical) and continuous (numerical) dynamics. They consist of a transition system, augmented with real-valued state variables that evolve according to a particular differential equation in every discrete state (mode). Conditions on the values of these variables may trigger discrete transitions (mode switching). Naturally, the verification of hybrid systems requires ingredients taken from the classical verification of transition systems, augmented with new special techniques for doing verification-like operations (successor computation) on the continuous dynamics. Early tools [11,1] focused on relatively simple continuous dynamics in each discrete state, where the derivative of the continuous variables does not depend on their values. For such “linear” hybrid automata, the computation of successors in the continuous domain can be realized by linear algebra. Nevertheless, it turned out that switching between such simple continuous modes one can easily construct undecidability gadgets and hence the *exact* verification of hybrid systems turned out to be a dead end.

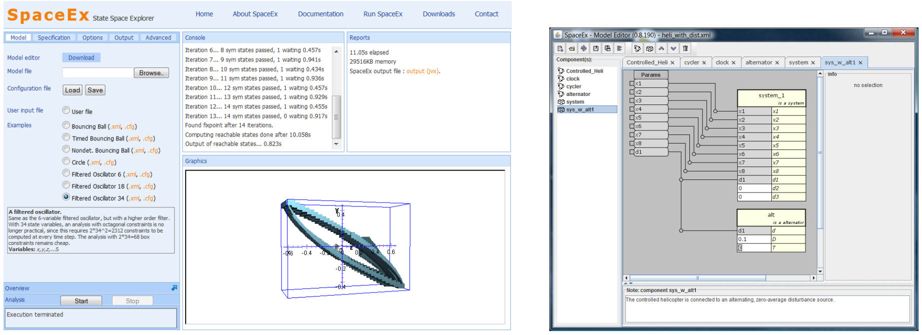
The second wave of hybrid verification tools [6,3,12] indeed abandoned exact computations and focused more on computing *approximations* of the reachable states for systems admitting less trivial continuous dynamics. Such techniques and tools could handle hybrid systems with continuous dynamics defined by linear differential equations with inputs. However, the size of systems that could be handled was modest, typically very few continuous state variables. Another thread in hybrid verification focuses on systems with a very large discrete state-space with very few continuous variables. Typically such models are aimed to verify the code of computerized control systems, with a very modest modeling of the external environment. The major preoccupation in these efforts is in combining the continuous part with techniques, such as BDD or SAT, for handling large discrete state-spaces [7,16].

In the last couple of years, an important breakthrough has been achieved in the reachability computation of the continuous part. A new algorithm based on a "lazy" representation of the reachable sets, first developed for the special case where reachable states are represented by Zonotopes [10] and then extended and crystalized using the more general representation via *support functions* [14,13,15], has dramatically increased the scope of linear systems that can be verified to several hundreds of state variables. Similar scale improvements have been reported recently on complementary approaches for handling linear systems [4]. Moreover, significant advances have been made on applying these linear techniques to nonlinear systems via "hybridization" (approximating a nonlinear system by a piecewise-affine one) [2], which have led to the verification of non-trivial nonlinear systems with about a dozen of state variables [8].

This paper is concerned with the transfer of such research achievements, typically obtained as theoretical results accompanied by a thesis-related prototype, into a robust and user-friendly toolset. It describes SpaceEx, a new extensible verification platform for hybrid systems, developed with systematic software engineering [9], implementing many of the above-mentioned developments, and featuring a web-based graphical user interface. As the reader will see, this transfer is not only about software engineering, modularity and user interface: it consists in improving and fine-tuning the algorithms to make them applicable to real-world problems. SpaceEx consists of three components:

- The *analysis core* is a command line program that takes a model file, a configuration file that specifies the initial states, the scenario and other options, and then analyzes the system and produces a series of output files.
- The *web interface*, shown in Fig. 1(a), is a graphical user interface with which one can comfortably specify initial states and other analysis parameters, run the analysis core, and visualize the output graphically. The web interface is browser-based, and accesses the analysis core via a web server, which may be running remotely or locally on a virtual machine.
- The *model editor*, shown in Fig. 1(b), is a graphical editor for creating models of complex hybrid systems out of nested components.

In this paper, we describe an efficient and scalable reachability algorithm, which builds on the one in [15], adapted specifically for maximum scalability. We present its extension to variable time steps, and propose an improved approximation model, which drastically improves the accuracy of the algorithm.



A filtered oscillator.
Same as the 6-variable filtered oscillator, but with a higher order filter. With 34 state variables, an analysis with octagonal constraints is no longer practical, since this requires 2*34^2=2312 constraints to be computed at every time step. The analysis with 2*34=68 box constraints remains cheap.
Variables: *x,y,z,...S*

Execution terminated

```
Iteration 6... 8 sym states passed, 1 waiting 0.457s
Iteration 7... 9 sym states passed, 1 waiting 0.941s
Iteration 8... 10 sym states passed, 1 waiting 0.434s
Iteration 9... 11 sym states passed, 1 waiting 0.936s
Iteration 10... 12 sym states passed, 1 waiting 0.457s
Iteration 11... 13 sym states passed, 1 waiting 0.929s
Iteration 12... 14 sym states passed, 1 waiting 0.455s
Iteration 13... 14 sym states passed, 0 waiting 0.917s
Found fixpoint after 14 iterations.
Computing reachable states done after 10.058s
Output of reachable states... 0.823s
```

```
11.05s elapsed
29516KB memory
SpaceEx output file : output (jvx).
```

system_1	
x1	*x1*
x2	*x2*
x3	*x3*
x4	*x4*
x5	*x5*
x6	*x6*
x7	*x7*
x8	*x8*
d1	*d1*
0	*d2*
0	*d3*

alt	
d1	*d*
0.1	*D*
3	*T*

The controlled helicopter is connected to an alternating, zero-average disturbance source.

(a) Web interface

(b) Model editor

Fig. 1. Graphical user interfaces of the SpaceEx platform

Experimental results demonstrate the scalability of the algorithm and the performance of the tool. Unlike in classical verification there are no established benchmarks and reference tools for comparing high-dimensional hybrid reachability. We know of no tool that has been reported to treat systems of the dimensions in this paper. The examples used in this paper as well as the tool itself are available at `http://spaceex.imag.fr`.

The paper is structured as follows. Section 2 recalls hybrid automata, the basic reachability algorithm and data structures used in SpaceEx. Section 3 describes the variable time step algorithm and the new approximation model used to compute time elapse successor states. The computation of successor states of discrete transitions is presented in Sect. 4. Experimental results based on our implementation in SpaceEx are provided in Sect. 5, followed by some conclusions in Sect. 6. The proofs for this paper are available as an appendix at `http://www-verimag.imag.fr/~frehse/cav2011_appendix.pdf`.

2 Reachability of Hybrid Systems

2.1 Hybrid Automata

We model the interaction of discrete events and continuous, time-driven dynamics with a *hybrid automaton* [1]. A hybrid automaton $H = (\mathit{Loc}, \mathit{Var}, \mathit{Lab}, \mathit{Inv}, \mathit{Flow}, \mathit{Trans}, \mathit{Init})$ consists of a labeled graph that encodes the nondeterministic evolution of a finite set of continuous *variables* Var over time. In this paper, we associate each of the n variables with a dimension in $\mathbb{R}^n$. A vertex $l \in \mathit{Loc}$ of the graph is called a *location*. A *state* is a pair $(l, x) \in \mathit{Loc} \times \mathbb{R}^n$. In every state (l, x), the time-driven evolution of the continuous variables is given by the set of derivatives $\mathit{Flow}(l, x) \subseteq \mathbb{R}^n$. The edges of the graph are called *discrete transitions*. A transition $(l, \alpha, \mathit{Guard}, \mathit{Asgn}, l') \in \mathit{Trans}$, with label $\alpha \in \mathit{Lab}$ allows the system to jump from location l to location l', instantaneously modifying the values of the variables. A state (l, x) can jump to (l', x') according to the *guard* $\mathit{Guard} \subseteq \mathbb{R}^n$ and the *assignment* $\mathit{Asgn}(x) \subseteq \mathbb{R}^n$, i.e., if $x \in \mathit{Guard}$ and $x' \in \mathit{Asgn}(x)$. The system may only remain in a location l as long as the state is inside the *invariant* $\mathit{Inv}(l) \subseteq \mathbb{R}^n$. All behavior originates from the *initial states* $\mathit{Init} \subseteq \mathit{Loc} \times \mathbb{R}^n$. In this paper, we consider $\mathit{Flow}(l)$ to be a continuous dynamics of the form

$$\dot{x}(t) = Ax(t) + u(t), \qquad u(t) \in \mathcal{U}, \tag{1}$$

where $x(t) \in \mathbb{R}^n$, A is a real-valued $n \times n$ matrix and $\mathcal{U} \subseteq \mathbb{R}^n$ is a closed and bounded convex set. Transition assignments Asgn are of the form

$$x' = Rx + w, \qquad w \in \mathcal{W}, \tag{2}$$

where R is a real-valued $n \times n$ matrix, and $\mathcal{W} \subseteq \mathbb{R}^n$ is a closed and bounded convex set of non-deterministic inputs.

An *execution* of the automaton is a sequence of discrete jumps and pieces of continuous trajectories according to its dynamics, and originates in one of the initial states. A state is *reachable* if an execution leads to it. We are concerned with computing the set of states that are reachable.

2.2 High-level Reachability Algorithm

Our reachability algorithm is a classical fixed-point computation that operates on *symbolic states.* A symbolic state is a pair (l, Ω), where l is a location and Ω is a convex continuous set. For a set of symbolic states $\mathcal{R}$, let the *discrete post-operator* $\mathrm{post}_d(\mathcal{R})$ be the set of states reachable by a discrete transition from $\mathcal{R}$, and the *continuous post-operator* $\mathrm{post}_c(\mathcal{R})$ be the set of states reachable from $\mathcal{R}$ by letting an arbitrary amount of time elapse. The set of reachable states is the fixed-point of the sequence $\mathcal{R}_0 = \mathrm{post}_c(\mathit{Init})$,

$$\mathcal{R}_{k+1} := \mathcal{R}_k \cup \mathrm{post}_c(\mathrm{post}_d(\mathcal{R}_k)). \tag{3}$$

In the following section we present how we represent convex continuous sets such that the post operators can be computed efficiently. The post operators themselves are presented in Sect. 3 and Sect. 4.

2.3 Support Functions and Template Polyhedra

Computing the image of the reachability post-operators for continuous sets is hard, in particular for the time elapse operator post_c. In [15], an efficient algorithm was proposed that uses support functions to represent convex continuous sets. The *support function* of a closed and bounded continuous set $\mathcal{S} \subseteq \mathbb{R}^n$ with respect to a direction vector $\ell \in \mathbb{R}^n$ is [5]

$$\rho(\ell, \mathcal{S}) = \max_{x \in \mathcal{S}} \ell \cdot x.$$

The support function of a convex set $\mathcal{S}$ is an exact representation of the set, which is illustrated by the fact that $\mathcal{S} = \bigcap_{\ell \in \mathbb{R}^n} \{x \mid \ell \cdot x \leq \rho(\ell, \mathcal{S})\}$. Representing a convex set by its support function has the benefit that the majority of the set operations in our algorithm can be implemented very efficiently:

- *linear map*: For a map $M \in \mathbb{R}^n \times \mathbb{R}^n$, $\rho(\ell, M\mathcal{S}) = \rho(M^\top \ell, \mathcal{S})$.
- *Minkowski sum*: For sets $\mathcal{S}_1$, $\mathcal{S}_2$, $\rho(\ell, \mathcal{S}_1 \oplus \mathcal{S}_2) = \rho(\ell, \mathcal{S}_1) + \rho(\ell, \mathcal{S}_2)$.
- *convex hull*: For sets $\mathcal{S}_1$, $\mathcal{S}_2$, $\rho(\ell, \mathrm{CH}(\mathcal{S}_1, \mathcal{S}_2)) = \max(\rho(\ell, \mathcal{S}_1), \rho(\ell, \mathcal{S}_2))$.

However, our algorithm requires two more operations, for which support functions are not efficient: intersection and deciding containment. For these operations, we use another set representation, *template polyhedra*, which are polyhedra with faces whose normal vectors are given a priori. Given a set $D = \{\ell_1, \ldots, \ell_m\}$ of vectors in $\mathbb{R}^n$ called *template directions*, a template polyhedron $\mathcal{P}_D \subseteq \mathbb{R}^n$ is a polyhedron for which there exist coefficients $b_1, \ldots, b_m \in \mathbb{R}$ such that

$$\mathcal{P}_D = \big\{x \in \mathbb{R}^n \mid \bigwedge_{\ell_i \in D} \ell_i \cdot x \leq b_i\big\}.$$

A template polyhedron can be represented by its coefficients b_i, which is particularly useful when working with sets of template polyhedra. In order to go from a support function representation of a set $\mathcal{S}$ to a template polyhedron, we use its *template hull* $\mathrm{TH}_D(\mathcal{S})$, which is the template polyhedron defined by coefficients $b_i = \rho(\ell_i, \mathcal{S})$. The support function of a polyhedron can be computed efficiently for a given direction ℓ using linear programming. In this paper, we consider:

- $2n$ *box* directions: $x_i = \pm 1$, $x_k = 0$ for $k \neq i$;
- $2n^2$ *octagonal* directions: $x_i = \pm 1$, $x_j = \pm 1$, $x_k = 0$ for $k \neq i$ and $k \neq j$;
- m *uniform* directions (as evenly as possible distributed over the unit sphere).

However, the algorithm supports a more general choice of directions, which remains to be investigated.

The use of both support functions and template hulls is justified by the fact that they are efficient for different operations, and both set representations are present in our implementation. Support functions are an exact and complete representation of convex sets – implemented as function objects, they can compute values for any direction. With template hulls, the directions D are fixed once and for all at the time of construction, and information for other directions is lost. By switching representations only when necessary (data-dependently) we remain as precise as possible.

3 Variable Time-Step Flowpipe Computation

We consider the affine continuous dynamics in (1), $\dot{x}(t) = Ax(t) + u(t)$, $u(t) \in \mathcal{U}$, where $x(0) \in \mathcal{X}_0$, and $\mathcal{U} \subseteq \mathbb{R}^n$ is a set of nondeterministic inputs. Let $\mathrm{Reach}_{t_1,t_2}(\mathcal{X})$ denote the reachable states starting from the set $\mathcal{X}$ with input set $\mathcal{U}$ in the time interval $[t_1, t_2]$,

$$\mathrm{Reach}_{t_1,t_2}(\mathcal{X}) = \{x(\tau) \mid t_1 \leq \tau \leq t_2, x(0) \in \mathcal{X}, x(t) \text{ satisfies (1)}\}. \tag{4}$$

We compute a *flowpipe*, a sequence of continuous sets $\Omega_0, \ldots, \Omega_{N-1}$ that covers the reachable states up to time T (N depends on the chosen time steps).

Before we present the actual algorithm, we discuss how we take into account the invariant of the corresponding location of the hybrid automaton. In our implementation, we test at the k-th step whether Ω_k is entirely outside of the invariant, and stop the sequence once this is the case. Then we intersect the invariant with the computed Ω_k (see Sect. 4 for a discussion of the intersection operation). Note that this procedure may produce an overapproximation, as the invariant may block some of the trajectories starting in $\mathcal{X}_0$ without blocking them all. We include the invariant face normals automatically in the template directions, so the result is usually of satisfactory precision. A variation of our algorithm with proper handling of invariants is presented in [14].

We now describe the variable time step algorithm. Given arbitrary time steps $\delta_0, \delta_1, \ldots$, we construct the sequence Ω_k that covers the set of reachable states. As we will show, each set Ω_k covers the reachable states in the time interval $[t_k, t_{k+1}]$, where $t_k = \sum_{i=0}^{k-1} \delta_i$. The algorithm is based on two functions $\Omega_{[0,\delta]}$ and Ψ_δ, called *approximation models.* They overapproximate the reachable states over a time interval $[0, \delta]$ as a function of $\delta, \mathcal{X}_0$, and $\mathcal{U}$:

$$\mathrm{Reach}_{0,\delta}(\mathcal{X}_0) \subseteq \Omega_{[0,\delta]}(\mathcal{X}_0, \mathcal{U}), \qquad \mathrm{Reach}_{\delta,\delta}(\{0\}) \subseteq \Psi_\delta(\mathcal{U}). \tag{5}$$

Each Ω_k is constructed by computing $\Omega_{[0,\delta_k]}$, which covers $\mathrm{Reach}_{0,\delta_k}(\mathcal{X}_0)$, and then shifting this set forward in time so that it covers $\mathrm{Reach}_{t_k,t_k+\delta_k}(\mathcal{X}_0)$. This is possible due to the following well-known property:

Lemma 1. $\mathrm{Reach}_{t_k,t_k+\delta_k}(\mathcal{X}) = e^{At_k}\mathrm{Reach}_{0,\delta_k}(\mathcal{X}) \oplus \mathrm{Reach}_{t_k,t_k}(\{0\})$.

It therefore suffices to apply the linear map e^{At_k} (a constant matrix) to $\Omega_{[0,\delta_k]}$, and to then add $\mathrm{Reach}_{t_k,t_k}(\{0\})$, which captures the influence of the inputs $\mathcal{U}$ up to time t_k. We overapproximate this summand with a sequence Ψ_k defined below:

$$\mathrm{Reach}_{t_k,t_k}(\{0\}) \subseteq \Psi_k.$$

Given approximation models $\Omega_{[0,\delta]}$ and Ψ_δ, we compute the sequence Ω_k as follows, with $\Psi_0 = \{0\}$:

$$\begin{aligned} \Psi_{k+1} &= \Psi_k \oplus e^{At_k}\Psi_{\delta_k}(\mathcal{U}), \\ \Omega_k &= e^{At_k}\Omega_{[0,\delta_k]}(\mathcal{X}_0, \mathcal{U}) \oplus \Psi_k \end{aligned} \tag{6}$$

Now we formally state the main result of this section: Given the approximation models, which we will define in the next section, the sequence Ω_k covers the reachable set.

Proposition 1. *Given a sequence of time steps $\delta_0, \ldots, \delta_{N-1}$ with $\sum_{i=0}^{N-1} \delta_i = T$, the sequence Ω_k defined by (6) satisfies*

$$\mathrm{Reach}_{0,T}(\mathcal{X}_0) \subseteq \bigcup_{k=0}^{N-1} \Omega_k. \tag{7}$$

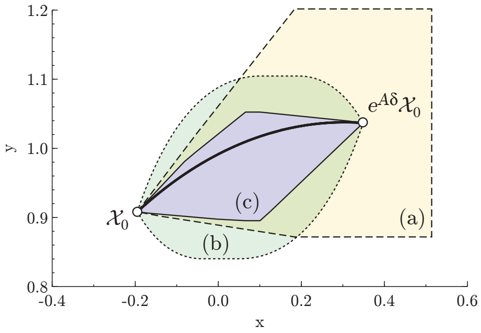


Fig. 2. Different approximation models for a segment of a circular trajectory, computed by SpaceEx: (a) using a first-order approximation of the ODE [15], (b) using a first-order approximation of the error of the linear interpolation between the states at time $t = 0$ and at $t = \delta$ [13], (c) the new model, which intersects a first-order approximation of the interpolation error going forward in time from $t = 0$ with one that goes backward from $t = \delta$.

Representing the sets Ψ_k and Ω_k by their support function, the operators used in (6) – linear map and Minkowski sum – can be computed efficiently as discussed in Sect. 2.3.

In the next section, we present the new approximation model which we use to compute Ω_k as in (6). In Sect 3.2, we provide direction-wise error bounds on this computation, and discuss how to adapt the time steps in order to guarantee given error bounds.

3.1 Approximation Model

The approximation quality of the sequence Ω_k evidently hinges on the quality of the approximation model. In [15], an approximation model was proposed that uses the norms of A, $\mathcal{X}_0$ and $\mathcal{U}$ to bound the error of a first-order approximation of the solution of the state equations, see Fig. 2 for an illustration. If this allows one to establish an asymptotically optimal error over a given time interval, it is sometimes overly conservative in practice. In this section, we propose a new model, which is a strict improvement over the method in [15] if the norm used is the infinity norm. For other norms, it is possible to find cases for which the resulting sets are incomparable, but we have yet to find a pratical case where our approximation is worse. Similar to a model proposed in [13], it is based on a first-order approximation of the interpolation error between the reachable set at time 0 and at time δ. On top of that, it combines an approximation that goes forward in time with one that goes backward in time in order to further improve the accuracy.

Before presenting the model, we introduce the following notations. The *symmetric interval hull* of a set S, denoted $\boxdot(S)$, is $\boxdot(S) = [-\overline{|x_1|}; \overline{|x_1|}] \times \ldots \times [-\overline{|x_d|}; \overline{|x_d|}]$ where for all i, $\overline{|x_i|} = \sup\{|x_i| \mid x \in S\}$. Let $M = (m_{i,j})$ be a matrix, and $v = (v_i)$ a vector. We define as $|M|$ and $|v|$ the absolute values of M and v respectively, i.e., $|M| = (|m_{i,j}|)$ and $|v| = (|v_i|)$. These absolute values allow us to bound matrix-vector operations (component wise) without taking their norm. The approximation model uses the following transformation matrices:

$$\varPhi_1(A,\delta) = \sum_{i=0}^{\infty} \frac{\delta^{i+1}}{(i+1)!} A^i, \qquad \varPhi_2(A,\delta) = \sum_{i=0}^{\infty} \frac{\delta^{i+2}}{(i+2)!} A^i. \tag{8}$$

If A is invertible, $\varPhi_1$ and $\varPhi_2$ can be computed as $\varPhi_1(A,\delta) = A^{-1}\left(e^{\delta A} - I\right)$, $\varPhi_2(A,\delta) = A^{-2}\left(e^{\delta A} - I - \delta A\right)$. Otherwise, they can be computed as submatrices of the block matrix

$$\begin{pmatrix} e^{A\delta} & \varPhi_1(A,\delta) & \varPhi_2(A,\delta) \\ 0 & I & I\delta \\ 0 & 0 & I \end{pmatrix} = \exp \begin{pmatrix} A\delta & I\delta & 0 \\ 0 & 0 & I\delta \\ 0 & 0 & 0 \end{pmatrix}.$$

We can now use these operators to obtain a precise over-approximation of $\mathrm{Reach}_{0,\delta}(\mathcal{X}_0)$ and $\mathrm{Reach}_{\delta,\delta}(\{0\})$. We start with a first-order approximation of the latter :

Lemma 2. *Let $\varPsi_\delta(\mathcal{U})$ be the set defined by*

$$\varPsi_\delta(\mathcal{U}) = \delta\mathcal{U} \oplus \mathcal{E}_\varPsi(\mathcal{U},\delta), \tag{9}$$

$$\mathcal{E}_\varPsi(\mathcal{U},\delta) = \boxdot\big(\varPhi_2(|A|,\delta) \boxdot (A\mathcal{U})\big). \tag{10}$$

Then, $\mathrm{Reach}_{\delta,\delta}(0) \subseteq \varPsi_\delta(\mathcal{U})$*.*

We now have discretized the differential equation, but we still have to over-approximate $\mathrm{Reach}_{0,\delta}(\mathcal{X}_0)$ with a function $\varOmega_{[0,\delta]}(\mathcal{X}_0,\mathcal{U})$. Our starting point is a linear interpolation between $\mathrm{Reach}_{0,0}$ and $\mathrm{Reach}_{\delta,\delta}$ using a parameter $\lambda = t/\delta$ representing normalized time. For each point in time t, $\mathrm{Reach}_{t,t} = \mathrm{Reach}_{\lambda\delta,\lambda\delta}$ is a convex set, for which we construct an over-approximation $\varOmega_\lambda$. Our over-approximation for the time interval $[0,\delta]$ is then the convex hull of all $\varOmega_\lambda$ over $\lambda \in [0,1]$. Using a forward, respectively backward, interpolation leads to an error term proportional to λ, respectively $1-\lambda$. Intersecting both approximations gives the following result:

Lemma 3. *Let $\lambda \in [0,1]$, and let $\varOmega_\lambda(\mathcal{X}_0,\mathcal{U},\delta)$ be the convex set defined by:*

$$\begin{aligned} \varOmega_\lambda(\mathcal{X}_0,\mathcal{U},\delta) &= (1-\lambda)\mathcal{X}_0 \oplus \lambda e^{\delta A}\mathcal{X}_0 \oplus \lambda\delta\mathcal{U} \\ &\qquad \oplus \big(\lambda\mathcal{E}_\varOmega^+(\mathcal{X}_0,\delta) \cap (1-\lambda)\mathcal{E}_\varOmega^-(\mathcal{X}_0,\delta)\big) \oplus \lambda^2\mathcal{E}_\varPsi(\mathcal{U},\delta), \ \textit{with} \\ \mathcal{E}_\varOmega^+(\mathcal{X}_0,\delta) &= \boxdot\left(\varPhi_2(|A|,\delta) \boxdot \left(A^2\mathcal{X}_0\right)\right), \\ \mathcal{E}_\varOmega^-(\mathcal{X}_0,\delta) &= \boxdot\left(\varPhi_2(|A|,\delta) \boxdot \left(A^2 e^{\delta A}\mathcal{X}_0\right)\right), \\ \mathcal{E}_\varPsi(\mathcal{U},\delta) &= \boxdot\left(\varPhi_2(|A|,\delta) \boxdot (A\mathcal{U})\right). \end{aligned}$$

Then $\mathrm{Reach}_{\lambda\delta,\lambda\delta}(\mathcal{X}_0) \subseteq \varOmega_\lambda(\mathcal{X}_0,\mathcal{U},\delta)$*. If we define $\varOmega_{[0,\delta]}(\mathcal{X}_0,\mathcal{U})$ as:*

$$\varOmega_{[0,\delta]}(\mathcal{X}_0,\mathcal{U}) = \mathrm{CH}\big(\bigcup_{\lambda\in[0,1]} \varOmega_\lambda(\mathcal{X}_0,\mathcal{U},\delta)\big), \tag{11}$$

then $\mathrm{Reach}_{0,\delta}(\mathcal{X}_0) \subseteq \varOmega_{[0,\delta]}(\mathcal{X}_0,\mathcal{U})$*.*

$\Omega_{[0,\delta]}(\mathcal{X}_0, \mathcal{U})$, as defined above, might seem hard to represent. In fact, its support function is not much harder to compute than the one of $\mathcal{X}_0$ and $\mathcal{U}$. Let

$$\begin{aligned}\omega(\mathcal{X}_0, \mathcal{U}, \delta, \lambda) = (1-\lambda)\rho(\ell, \mathcal{X}_0) + \lambda\rho((e^{\delta A})^\top \ell, \mathcal{X}_0) + \lambda\delta\rho(\ell, \mathcal{U}) \\ + \rho\left(\ell, \lambda\mathcal{E}_\Omega^+ \cap (1-\lambda)\mathcal{E}_\Omega^-\right) + \lambda^2\rho(\ell, \mathcal{E}_\Psi). \quad (12)\end{aligned}$$

Since the signs of λ, $(1-\lambda)$, and λ^2 do not change on $[0,1]$, we have:

$$\rho(\ell, \Omega_{[0,\delta]}(\mathcal{X}_0, \mathcal{U})) = \max_{\lambda \in [0,1]} \omega(\mathcal{X}_0, \mathcal{U}, \delta, \lambda). \quad (13)$$

The support function of $\lambda\mathcal{E}_\Omega^+ \cap (1-\lambda)\mathcal{E}_\Omega^-$ can easily be expressed as a piecewise linear function of λ. For any λ, this set is a centrally symmetric box and its support function is:

$$\rho(\ell, \lambda\mathcal{E}_\Omega^+ \cap (1-\lambda)\mathcal{E}_\Omega^-) = \sum_{i=1}^{d} \min(\lambda e_i^+, (1-\lambda)e_i^-)|l_i|,$$

where e^+ and e^- are such that $\rho(\ell, \mathcal{E}_\Omega^+) = e^+ \cdot |l|$ and $\rho(\ell, \mathcal{E}_\Omega^-) = e^- \cdot |l|$. Thus we only have to maximize a piecewise quadratic function in one variable on $[0,1]$ after the evaluation of the support function of the sets involved.

3.2 Time Step Adaptation with Error Bounds

Time steps are generally hard to choose, and their value is rarely chosen in itself, but according to an expected precision. In order to efficiently choose our variable time step, we must be able to evaluate the error introduced by time discretization. In our error calculation, we do not bound the error introduced at each step, but the overall error introduced since the beginning of the current continuous evolution. We must take into account the accumulation of errors, not done carefully we might exhaust our error tolerance before the end of the computation and become unable to advance in time without exceeding it.

Our choice of error bound $\varepsilon(\ell)$ is the difference between the computed support function and the support function of the reachable set at time t. For each set Ω_k we define

$$\varepsilon_{\Omega_k}(\ell) = \rho\left(\ell, \Omega_k\right) - \rho\left(\ell, \mathrm{Reach}_{t_k, t_{k+1}}(\mathcal{X}_0)\right) \quad (14)$$

The total approximation error is then

$$\varepsilon(\ell) = \max_{k \in 0, \ldots, N-1} \varepsilon_{\Omega_k}(\ell).$$

Note that $\mathrm{Reach}_{t_k, t_{k+1}}(\mathcal{X}_0)$ is generally not a convex set, and by overapproximating it with the convex set Ω_k we incur an additional error that is not captured by $\varepsilon_{\Omega_k}(\ell)$. The bound $\varepsilon(\ell)$ allows us to decide whether or not the reachable set satisfies a linear constraint $\ell \cdot x \leq b$ (e.g., whether the states surpass a certain threshold) with an uncertainty margin of $\varepsilon(\ell)$.

To compute $\varepsilon_{\Omega_k}(\ell)$, we must take into account the error for $\Psi_\delta(\mathcal{U})$ defined as in Lemma 2 and $\Omega_{[0,\delta]}(\mathcal{X}_0,\mathcal{U})$ defined as in Lemma 3. Let

$$\varepsilon_{\Psi_\delta(\mathcal{U})}(\ell) = \rho\left(\ell, \Psi_\delta(\mathcal{U})\right) - \rho\left(\ell, \mathrm{Reach}_{\delta,\delta}(\{0\})\right), \tag{15}$$

$$\varepsilon_{\Omega_{[0,\delta]}(\mathcal{X}_0,\mathcal{U})}(\ell) = \rho(l, \Omega_{[0,\delta]}(\mathcal{X}_0,\mathcal{U})) - \rho(l, \mathrm{Reach}_{0,\delta}(\mathcal{X}_0)). \tag{16}$$

Lemma 4. *For any ℓ in $\mathbb{R}^n$:*

$$\varepsilon_{\Psi_\delta(\mathcal{U})}(\ell) \leq \rho(\ell, \mathcal{E}_\Psi(\mathcal{U},\delta)) + \rho\left(\ell, -A\Phi_2(A,\delta)\mathcal{U}\right) \tag{17}$$

$$\begin{aligned}\varepsilon_{\Omega_{[0,\delta]}(\mathcal{X}_0,\mathcal{U})}(\ell) \leq \max_{\lambda\in[0,1]} \Big\{ &\rho\Big(l, \big(\lambda\mathcal{E}_\Omega^+(\mathcal{X}_0,\delta) \cap (1-\lambda)\mathcal{E}_\Omega^-(\mathcal{X}_0,\delta)\big)\Big) \\ &+\lambda^2\rho(l, \mathcal{E}_\Psi(\mathcal{U},\delta)) + \lambda\rho\left(\ell, -A\Phi_2(A,\delta)\mathcal{U}\right)\Big\}.\end{aligned} \tag{18}$$

Computing the support function of Ω_k as defined by (6) and applying the above lemmas as well as Lemma 1, we obtain $\varepsilon_{\Omega_k}(\ell)$ as follows.

$$\begin{aligned}\varepsilon_{\Psi_{k+1}}(\ell) &= \varepsilon_{\Psi_k}(\ell) + \varepsilon_{\Psi_{\delta_k}(\mathcal{U})}({e^{At_k}}^\top\ell), \\ \varepsilon_{\Omega_k}(\ell) &= \varepsilon_{\Omega_{[0,\delta_k]}(\mathcal{X}_0,\mathcal{U})}({e^{At_k}}^\top\ell) + \varepsilon_{\Psi_k}(\ell).\end{aligned} \tag{19}$$

Given the above error bounds, one can adapt the time steps during the computation of the sequence such that $\varepsilon(\ell)$ is kept arbitrarily small. The problem lies in the error $\varepsilon_{\Psi_k}(\ell)$, which accumulates with k, so that an a-posteriori refinement would require the whole squence to be recomputed. We therefore use the following simple heuristic to compute the sequence of $\rho(\ell, \Omega_k)$ for a given template direction ℓ such that $\varepsilon(\ell) \leq \hat{\varepsilon}$ for a given error bound $\hat{\varepsilon} \in \mathbb{R}^{>0}$. Instead of computing $\rho(\ell, \Omega_k)$ directly, we first compute the whole sequence $\rho(\ell, \Psi_k)$, then the sequence $\rho({e^{At_k}}^\top\ell, \Omega_{[0,\delta_k]}(\mathcal{X}_0,\mathcal{U}))$, and only then combine them to get the sequence $\rho(\ell, \Omega_k)$. This separation allows us to choose a separate time step for each sequence, adapting the error as necessary. Additionally, by computing the sequences one after another, the last one can pick up the slack in the error bound of the first sequence.

In the following we suppose that $\hat{\varepsilon}$ is distributed a-priori on both sequences, so that we have $\hat{\varepsilon}_\Omega$ and $\hat{\varepsilon}_\Psi$ in $\mathbb{R}^{>0}$ with $\hat{\varepsilon} = \hat{\varepsilon}_\Omega + \hat{\varepsilon}_\Psi$. This distribution can be established, e.g., by a prior coarse-grained run with a large error bound, or by a run with large fixed time steps. Because the error of Ψ_k accumulates with k, it is chosen (somewhat arbitrarily) to remain below a linearly increasing bound. The error of $e^{At_k}\Omega_{[0,\delta_k]}(\mathcal{X}_0,\mathcal{U})$ does not depend on previous computation steps, so it can be adapted on the fly to meet the required bound. We proceed as follows:

1. Compute $\rho(\ell, \Psi_k(\mathcal{U}))$ such that $\varepsilon_{\Psi_k}(\ell) \leq \frac{t_k^\Psi}{T}\hat{\varepsilon}_\Psi$. At each step k, we must find a δ_k^Ψ, ideally the biggest, such that:

$$\varepsilon_{\Psi_k}(\ell) + \varepsilon_{\Psi_{\delta_k^\Psi}(\mathcal{U})}({e^{At_k^\Psi}}^\top\ell) \leq \frac{t_k^\Psi + \delta_k^\Psi}{T}\hat{\varepsilon}_\Psi$$

Finding δ_k^{Ψ} is possible because $\varepsilon_{\Psi_\delta(\mathcal{U})}(e^{At^\top}\ell) = O(\delta^2)$. First, we fix an initial time step δ_{-1}^{Ψ}. Then, at each step k, we start a dichotomic search from δ_{k-1}^{Ψ} along the δ for which sets and matrices involved in the computation of $\Psi_\delta(\mathcal{U})$ have already been evaluated, trying new values only when necessary.

2. Compute $\rho(e^{At_k^{\Omega}{}^\top}\ell, \Omega_{[0,\delta_k^{\Omega}]}(\mathcal{X}_0,\mathcal{U}))$ such that

$$\varepsilon_{\Omega_{[0,\delta_k^{\Omega}]}(\mathcal{X}_0,\mathcal{U})}(e^{At_k^{\Omega}{}^\top}\ell) + \varepsilon_{\Psi_{i(k)}}(\ell) = \varepsilon_{\Omega_k}(\ell) \leq \hat{\varepsilon}$$

   where $i(k)$ is such that $t_{i(k)-1}^{\Psi} < t_k^{\Omega} \leq t_{i(k)}^{\Psi}$. This can be done with a dichotomic search over the sequence of t_k^{Ψ} already computed for $\rho(\ell, \Psi_k(\mathcal{U}))$. If there is a k such that $\delta_{i(k)}^{\Psi}$ must be further refined, then for the newly introduced indices k_j, we have $i(k_j) = i(k)$.

To combine the above two sequences into $\rho(\ell, \Omega_k)$, we use the sequence of time steps δ_k^{Ω}. If this introduces new times t_k^{Ω} in the sequence of t_k^{Ψ}, we can compute the missing values for $\rho(\ell, \Psi_k(\mathcal{U}))$ by starting from $t_{i(k)-1}^{\Psi}$. This does not trigger the recomputation of $\rho(\ell, \Psi_k(\mathcal{U}))$ for other time points since the sequence $\varepsilon_{\Psi_k}(\ell)$ is increasing.

4 Computing Transition Successors

Each flow-pipe that is created by the time elapse step is passed to the computation of transition successors. States that take the transition must satisfy the guard, are then mapped according to the assignment and the result must satisfy the invariant of the target location. Consider a transition T with guard $\mathcal{G}$, assignment $Asgn$, and whose target location has the invariant $\mathcal{I}^+$. Let $Post_{Asgn}(\mathcal{X})$ be the set of states that result from mapping a continuous set $\mathcal{X}$ according to $Asgn$. Then for a set of states $\mathcal{X}$, the set of successor states is given by

$$\mathrm{post}_d(T, \mathcal{X}) = Post_{Asgn}(\mathcal{X} \cap \mathcal{G}) \cap \mathcal{I}^+.$$

We now discuss how the operations for this image computation, intersection and assignment, can be carried out efficiently. Then we present a method to decrease the number of convex sets produced by the successor computation so that an exponential blowup in the number of sets can be avoided.

Intersection Since intersection with support functions is hard, we compute intersection on the template hull, say $\mathcal{P}_D = \mathrm{TH}_D(\mathcal{X})$. If $\mathcal{X}$ is a set of convex sets, the intersection is performed separately on each convex set. If $\mathcal{G}$ is a polyhedron in constraint form whose constraint normals are template directions, then this operation can be carried out very efficiently by taking the minimum of the template coefficients:

$$\mathcal{P}_D \cap \mathcal{G} = \big\{x \in \mathbb{R}^n \mid \bigwedge_{\ell_i \in D} \ell_i \cdot x \leq \min(b_i^{\mathcal{X}}, b_i^{\mathcal{G}})\big\}.$$

Assignment Recall that according to (2) transition assignments are of the form $x' = Rx + w, w \in \mathcal{W}$, where $\mathcal{W} \subseteq \mathbb{R}^n$ is a convex set of non-deterministic inputs. In the general case, the assignment operator is therefore

$$Post_{Asgn}(\mathcal{X}) = R\mathcal{X} \oplus \mathcal{W},$$

and can be computed efficiently using support functions. If the assignment is invertible and deterministic, i.e., R is invertible and $\mathcal{W} = \{w_0\}$ for some constant vector w_0, the exact image can be computed efficiently on the polyhedron.

Clustering Each flow-pipe consists of a possibly large number of convex sets that cover the actual trajectories. When we compute the successor states for a transition, each of these convex sets spawns its own flow-pipe in the next time elapse computation. This may multiply the number of sets with each iteration, leading to an explosion in the number of sets and slowing the analysis down to a stall. To avoid this effect and speed up the analysis, we apply what we call *clustering*. Given a hull operator, clustering reduces the number of sets by replacing groups of these sets with a single convex set, their hull. We use the following clustering algorithm for a given hull operator *HULL*. Let the *width* of $\mathcal{P}_1, \dots, \mathcal{P}_z$ with respect to a direction $\ell \in D$ be

$$\delta_{\mathcal{P}_1,\dots,\mathcal{P}_z}(\ell) = \max_{i=1..z} \rho(\ell, \mathcal{P}_i) - \min_{i=1..z} \rho(\ell, \mathcal{P}_i). \tag{20}$$

Given $\mathcal{P}_1, \dots, \mathcal{P}_z$ and a *clustering factor* of $0 \leq c \leq 1$, the clustering algorithm produces a set of polyhedra $Q_1, \dots, Q_r$, $r \leq z$, as follows:

1. Let $i = 1$, $r = 1$, $Q_r := \mathcal{P}_i$.
2. While $i \leq z$ and $\forall \ell \in D : \delta_{Q_r,\mathcal{P}_i}(\ell) \leq c\delta_{\mathcal{P}_1,\dots,\mathcal{P}_z}(\ell)$,
   $Q_r := HULL(Q_r, \mathcal{P}_i)$, $i := i + 1$.
3. If $i \leq z$, let $r := r + 1$, $Q_r := \mathcal{P}_i$. Otherwise, stop.

We consider two hull operators: template hull, which is fast but very overapproximative, and convex hull, which is precise but slower. It can be advatageous to combine both, as illustrated by the following example:

Example 1. Consider the 8-variable filtered oscillator from Sect. 5.1. At the first discrete state change alone, 57 convex sets can take the transition. Without clustering, the computation is not feasible, as these sets would spawn 57 new flowpipes, and similarly for their successors. Template hull clustering with $c_{TH} = 0.3$ produces three sets and results in a total runtime of 11.5s. With $c_{TH} = 1$ it produces a single set and takes 3.36s, but with a large overapproximation. Convex hull clustering by itself with $c_{CH} = 1$ is very precise but takes 8.19s. Combining both with $c_{TH} = 0.3, c_{CH} = 1$ takes 3.64s without noticeable loss in accuracy.

5 Experimental Results

To demonstrate the scalability of our algorithm and the performance of the tool SpaceEx, we present the following experiments. The first system is a simple

Table 1. Performance results for the filtered oscillator benchmark, varying the number of variables in the system. The time step is $\delta = 0.05$, applying template hull clustering with $c_{TH} = 0.3$ followed by convex hull clustering with $c_{CH} = 1$. Indicated are the runtime, memory and iterations required to compute a fixed-point, and the largest error for any of the directions in any of the time steps

Variables	Time [s]	Mem. [MB]	Iter.	Error
Box constraints				
18	2,0	9,3	9	0,010
34	9,1	20,2	13	0,010
66	77,3	50,3	23	0,013
130	1185,6	194,3	39	0,030
198	7822,5	511,0	57	0,074

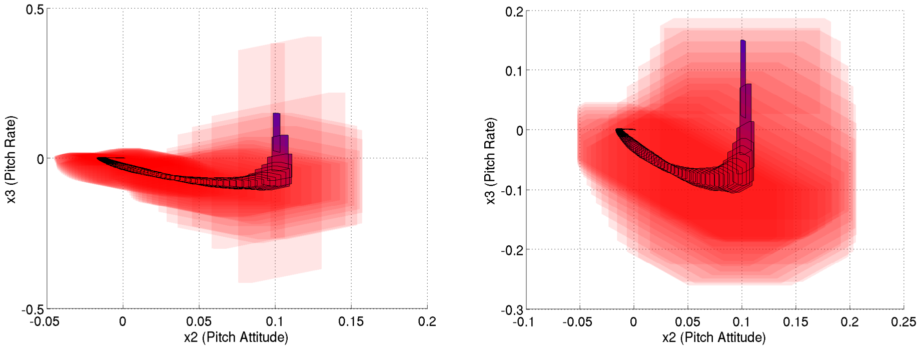
Variables	Time [s]	Mem. [MB]	Iter.	Error
Octagonal constraints				
2	0,7	11,8	6	0.009
4	1,4	11,8	6	0.025
6	4,7	13,3	6	0.025
10	33,0	23,0	7	0.025
18	538,4	67,9	10	0.025

parameterized system which we use to empirically measure the complexity of our algorithm. The second system is a multivariable continuous control system with complex, tightly coupled dynamics. It illustrates the faithfulness and accuracy of the continuous part of the algorithm. For lack of space we do not present the model equations, but instead refer the reader to the SpaceEx model files, which are available at `http://spaceex.imag.fr`.

5.1 Filtered Oscillator Benchmark

To measure the performance of our approach, we use a simple parameterized switched oscillator system. The complexity of the system is increased by adding a series of first-order filters to the output x of the oscillator. The filters smooth x, producing a signal z whose amplitude diminishes as the number of filters increases. Note that the dynamics are rather simple, as the filter variables are only weakly coupled with one another. The oscillator is an affine system with variables x, y that switches between two equilibria in order to maintain a stable oscillation, which together with k filters yields a parameterized system with $k + 2$ continuous variables and two locations. One location has the invariant $x \geq -1.4y$, the other $x \leq -1.4y$, and the guards consist of the boundaries of the invariants.

To empirically measure how the complexity depends on the n variables of the system and the m template directions, we run experiments varying just n, just m, and both. *Fixed n:* The average time for a successor computation (discrete followed by continuous) for the 6-variable system over m uniform directions, $m = 8$—256, shows an root mean sqare (RMS) tendency of $O(m^{1.7})$. *Fixed m:* The average time for a successor computation with 200 uniform directions with $n = 6$–16 shows an RMS tendency of $O(n)$. Table 1 shows the complete runtime of a fixed-point computation for box and uniform directions for the system with up to 198 variables. The RMS tendency is $O(n^{2.7})$ for box directions and $O(n^{4.7})$ for octagonal directions, which confirms the results for fixed n and fixed m.



(a) $\delta t = 0.05$ for both (b) forw: $\delta t = 0.005$, interpfb: $\delta t = 0.05$

Fig. 3. The reachable states of the 28-variable controlled helicopter system in the plane (x_2, x_3) (corresponding to roll attitude and roll rate), computed with octagonal constraints. We compare the new error scheme (interfb), shown in dark blue, with that of [13] (a) and of [15] (b), shown in bright red.

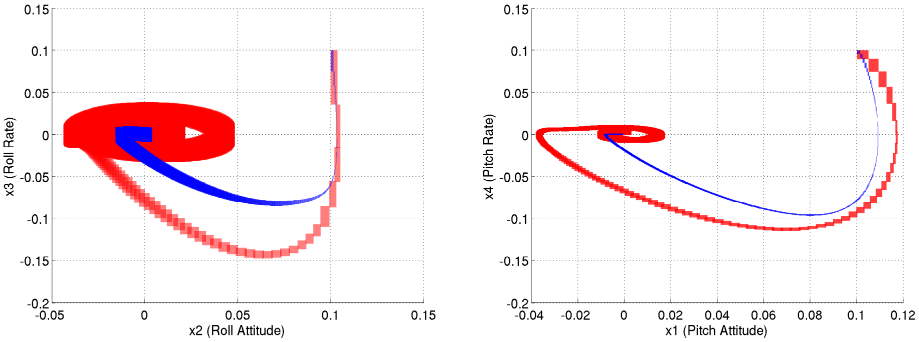
Table 2. Error models comparison with fixed step. The error model introduced in this paper (interpfb) clearly outperforms that proposed in [15] (forward) and the one proposed in [13] (interp). Memory is indicated in MBs.

	forward			interp			interpfb		
δt	Mem.	Time [s]	Error	Mem.	Time [s]	Error	Mem.	Time [s]	Error
0.05	9.44	1.48	9.67e+22	9.61	1.60	16.1	9.59	1.65	2.95
0.01	10.5	7.09	3.85e+5	10.5	7.60	0.191	10.5	8.16	0.178
5e-3	10.3	14.1	2.47e+3	10.2	15.2	4.37e-2	12.6	15.8	2.82e-2
1e-3	23.1	71.1	12.4	18.4	76.7	1.59e-3	18.5	78.7	1.07e-3
5e-4	27.9	142	2.56	27.9	155	3.89e-4	28.2	157	2.66e-4

5.2 Helicopter Controller

To measure the performance of our algorithm for complex dynamics, we analyze the helicopter controller from [17]. We analyze the controlled plant, a 28-dimensional continuous linear time-invariant (LTI) system. The plant is a small disturbance model of a helicopter, given as an 8-dimensional LTI system. The controller we examine is an $\mathcal{H}_\infty$ mixed-sensitivity design for rejecting atmospheric turbulence, given as a 20-dimensional LTI system. Figure 3 shows the increased accuracy of our new approximation model with respect to previous models. Tables 2 and 3 show the performance results with fixed time steps and with variable time steps, each for the different error models.

In [17], two different controllers are designed for the helicopter, one of which is specifically tuned for disturbance rejection. Letting the rotor collective be a nondeterministic input in the interval $[-1, 1]$, we compute the reachable states in 5s for one controller and in 14s for the other, as shown in Fig. 4.



(a) Roll stabilization (b) Pitch stabilization

Fig. 4. Comparison of two disturbance rejection models. Reachable sets with nondeterministic inputs for the helicopter example for the two disturbance rejection models compared in [17] ($T = 20$, method interfb and error tolerance of 0.1 for both). This confirms that the better disturbance rejection model proposed (in blue) actually stabilizes the system faster. However, in [17], this analysis was based only on several simulations.

Table 3. Variable step performance. The variable step implementation outperforms a fixed step scheme even in the *ideal case*, i.e., with the best error model and assuming we know in advance the optimal time step δt to satisfy the error bound. This is always true for the number of steps taken and the slightly higher computational time for some case is explained by frequent changes in choice of the time step.

	Ideal fixed step (interpfb)			var step (interp)		var step (interpfb)	
Err. req.	nb steps	δt used	Time [s]	nb steps	Time [s]	nb steps	time
1	1500	0.02	11.68	1475	12.0	974	8.359
0.1	3418	8.78e-3	26.6	4334	33.9	2943	31.2
0.01	11539	2.6e-3	90.3	14070	108	9785	77.9
1e-3	44978	6.67e-4	351	39152	301	27855	234
1e-4	101352	2.96e-4	902	85953	688	64315	811

6 Conclusions

The reachability of hybrid systems is recognized as be a hard problem, and research efforts to find a scalable approach have been going on for more than two decades. In this paper, we presented a variable time-step extension of a scalable time-elapse algorithm and proposed an improved, highly accurate approximation model for it. Together with techniques to efficiently compute transition successors that avoid the well-known problem of an explosion in the number of sets that the algorithm needs to propagate, we have implemented the algorithm in a new tool called SpaceEx. In our experiments, SpaceEx can handle hybrid systems with affine dynamics and nondeterministic inputs with more than 100 variables. Further research is needed to automatically find suitable template directions to increase the accuracy of the approach. SpaceEx and the examples from this paper are available at `http://spaceex.imag.fr`.

Acknowledgments This research was supported in part by the European Commission under grant INFSO-ICT-224249 and the U.S. National Science Foundation under grant number CCF-0926181.

References

1. 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*, 138(1):3–34, 1995.
2. E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Inf.*, 43(7):451–476, 2007.
3. E. Asarin, T. Dang, O. Maler, and O. Bournez. Approximate reachability analysis of piecewise-linear dynamical systems. In *HSCC'00*, LNCS. Springer, 2000.
4. E. Asarin, T. Dang, O. Maler, and R. Testylier. Using redundant constraints for refinement. In *Proc. ATVA'10*. Springer, 2010.
5. D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, 2003.
6. A. Chutinan and B. H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *HSCC'99*, LNCS, pages 76–90. Springer, 1999.
7. W. Damm, S. Disch, H. Hungar, S. Jacobs, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz. Exact state set representations in the verification of linear hybrid systems with large discrete state space. In *ATVA*, 2007.
8. T. Dang, C. Le Guernic, and O. Maler. Computing reachable states for nonlinear biological models. In *Proc. CMSB'09*, pages 126–141. Springer, 2009.
9. G. Frehse and R. Ray. Design principles for an extendable verification tool for hybrid systems. In *ADHS09*, 2009.
10. A. Girard, C. Le Guernic, and O. Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In *HSCC'06*. Springer, 2006.
11. T. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110 – 122, 1997.
12. A. Kurzhanski and P. Varaiya. Reachability analysis for uncertain systems—the ellipsoidal technique. *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Applications and Algorithms*, 9(3b):347–367, 2002.
13. C. Le Guernic. *Reachability analysis of hybrid systems with linear continuous dynamics*. PhD thesis, Université Grenoble 1 - Joseph Fourier, 2009.
14. C. Le Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 540–554. Springer, 2009.
15. C. Le Guernic and A. Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250 – 262, 2010.
16. C. Scholl, S. Disch, F. Pigorsch, and S. Kupferschmid. Computing optimized representations for non-convex polyhedra by detection and removal of redundant linear constraints. In *TACAS*, 2009.
17. S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons, 2005.