

Build Your Own Reachability Analyzer with CORA

(Tutorial)

Matthias Althoff

Technische Universität München, 85748 Garching, Germany

Abstract

The COntinuous Reachability Analyzer (CORA) is a collection of MATLAB classes for the formal verification of cyber-physical systems using reachability analysis. CORA integrates various vector and matrix set representations and operations on them as well as reachability algorithms of various dynamic system classes. The software is designed such that set representations can be exchanged without having to modify the code for reachability analysis. CORA is designed using the object oriented paradigm, such that users can safely use methods without concerning themselves with detailed information hidden inside the object. Since the toolbox is written in MATLAB, the installation and use is platform independent. We will demonstrate how easily CORA can be used by coding our own reachability analyzer based on CORA classes within this workshop. We will also demonstrate how to load SpaceEx files and how one can use already integrated reachability analysis approaches in CORA.

Contents

1	Introduction	2
2	Set Operations	3
2.1	Intervals	3
2.2	Zonotopes	4
2.3	Zonotope Bundles	5
2.4	Constrained Zonotopes	6
2.5	Polynomial Zonotopes	7
2.6	Further Set Representations	8
3	Linear Discrete-Time Systems	9
4	Linear Continuous-Time Systems	10
4.1	Homogeneous Solution for Points in Time	10
4.2	Homogeneous Solution for Time Intervals	11
4.3	Input Solution	13
4.4	Realization in CORA	13
5	Loading Models from SpaceEx	15
6	Nonlinear Systems	16
6.1	Main Principle	17
6.2	Quadrotor Example	18

7	Hybrid Systems	21
7.1	Hybrid Automata	21
7.2	Bouncing Ball Example	22
8	Further Continuous Dynamics	23
A	Troubleshooting	24

1 Introduction

The **CO**n**tinuous R**eachability **A**nalyzer (COR^A)¹ is a MATLAB toolbox for prototypical design of algorithms for reachability analysis. The toolbox is designed for various kinds of systems with purely continuous dynamics (linear systems, nonlinear systems, differential-algebraic systems, parameter-varying systems, etc.) and hybrid dynamics combining the aforementioned continuous dynamics with discrete dynamics. Let us denote the continuous part of the solution of a hybrid system for a given initial discrete state by $\chi(t; x_0, u(\cdot), p)$, where $t \in \mathbb{R}$ is the time, $x_0 \in \mathbb{R}^n$ is the continuous initial state, $u(t) \in \mathbb{R}^m$ is the system input at t , $u(\cdot)$ is the input trajectory, and $p \in \mathbb{R}^p$ is a parameter vector. The continuous reachable set at time $t = r$ can be defined for a set of initial states \mathcal{X}_0 , a set of input values $\mathcal{U}(t)$, and a set of parameter values \mathcal{P} , as

$$\mathcal{R}^e(r) = \left\{ \chi(r; x_0, u(\cdot), p) \in \mathbb{R}^n \mid x_0 \in \mathcal{X}_0, \forall t : u(t) \in \mathcal{U}(t), p \in \mathcal{P} \right\}.$$

COR^A solely supports over-approximative computation of reachable sets since (a) exact reachable sets cannot be computed for most system classes [10] and (b) over-approximative computations qualify for formal verification. Thus, COR^A computes over-approximations for particular points in time $\mathcal{R}(t) \supseteq \mathcal{R}^e(t)$ and for time intervals: $\mathcal{R}([t_0, t_f]) = \bigcup_{t \in [t_0, t_f]} \mathcal{R}(t)$.

COR^A enables one to construct one's own reachable set computation in a relatively short amount of time. This is achieved by the following design choices:

- COR^A is built for MATLAB, which is a script-based programming environment. Since the code does not have to be compiled, one can stop the program at any time and directly see the current values of variables. This makes it especially easy to understand the workings of the code and to debug new code.
- COR^A is an object-oriented toolbox that uses modularity, operator overloading, inheritance, and information hiding. One can safely use existing classes and just adapt classes one is interested in without redesigning the whole code. Operator overloading makes it possible to write formulas that look almost identical to the ones derived in scientific papers and thus reduce programming errors. Most of the information for each class is hidden and is not relevant to users of the toolbox. Most classes use identical methods so that set representations and dynamic systems can be effortlessly replaced.
- COR^A interfaces with the established toolbox MPT², which is also written in MATLAB. Results of COR^A can be easily transferred to this toolbox and vice versa. We are currently supporting version 2 and 3 of the MPT.

Of course, it is also possible to use COR^A as it is to conduct reachability analysis.

Please be aware of the fact that outcomes of reachability analysis heavily depend on the chosen parameters for the analysis. Improper choice of parameters can result in an unacceptable

¹<https://www6.in.tum.de/Main/SoftwareCOR>

²<http://control.ee.ethz.ch/~mpt/2/>

over-approximation although reasonable results could be achieved by using appropriate parameters. Thus, self-tuning of the parameters for reachability analysis is investigated as part of future work.

Since this manual focuses on the presentation of the capabilities of CORA, no other tools for reachability analysis of continuous and hybrid systems are reviewed. A list of related tools is presented in [3–5].

2 Set Operations

Efficient set operations are crucial for efficient reachability analysis. We will start with intervals as one of the simplest set representations and finish with polynomial zonotopes, which are a non-convex set representation.

2.1 Intervals

A real-valued interval $[x] = [\underline{x}, \bar{x}] = \{x \in \mathbb{R} | \underline{x} \leq x \leq \bar{x}\}$ is a connected subset of \mathbb{R} and can be specified by a left bound $\underline{x} \in \mathbb{R}$ and right bound $\bar{x} \in \mathbb{R}$, where $\underline{x} \leq \bar{x}$. A detailed description of how intervals are treated in CORA can be found in [4]. This class has a lot of methods, and almost all operators and functions defined in MATLAB can be applied to intervals, such as `abs`, `cos`, `power`, etc.

```

1 I1 = interval([0; -1], [3; 1]); % create interval I1
2 I2 = interval([-1; -1.5], [1; -0.5]); % create interval I2
3 Z1 = zonotope([1 1 1; 1 -1 1]); % create zonotope Z1
4
5 %% basic set operations
6 r = rad(I1) % obtain and display radius of I1
7 is_intersecting = isIntersecting(I1, Z1) % Z1 intersecting I1?
8 I3 = I1 & I2; % computes the intersection of I1 and I2
9 c = mid(I3) % returns and displays the center of I3
10
11 figure; hold on
12 plot(I1); % plot I1
13 plot(I2); % plot I2
14 plotFilled(I3, [1 2], [.6 .6 .6], 'EdgeColor', 'none'); % plot I3
15
16 %% mathematical functions
17 A_inf = [-1 -1; 0.5 -2]; % specify infimum
18 A_sup = [1 2; 1 -1]; % specify supremum
19 A = interval(A_inf, A_sup); % generate matrix
20 B_inf = [0 -1; 0 1]; % specify infimum
21 B_sup = [1 3; 0.5 1.2]; % specify supremum
22 B = interval(B_inf, B_sup); % generate matrix
23
24 A+B % addition
25 A*B % multiplication
26 A.*B % pointwise multiplication
27 A/interval(1,2) % division
28 A./B % pointwise division
29 A^3 % power function
30 sin(A) % sine function
31 sin(A(1,1)) + A(1,1)^2 - A(1,1)*B(1,1) % scalar combination of functions

```

32 `sin(A) + A^2 - A*B % matrix combination of functions`

2.2 Zonotopes

A zonotope is a point-symmetric set parameterized by a center $c \in \mathbb{R}^n$ and generators $g^{(i)} \in \mathbb{R}^n$:

$$\mathcal{Z} = \left\{ c + \sum_{i=1}^p \beta_i g^{(i)} \mid \beta_i \in [-1, 1] \right\}. \quad (1)$$

We write in short $\mathcal{Z} = (c, g^{(1)}, \dots, g^{(p)})$. A zonotope can be interpreted as the Minkowski addition of line segments $l^{(i)} = [-1, 1]g^{(i)}$ and is visualized step-by-step in a two-dimensional vector space in Fig. 1. Zonotopes are a compact way of representing sets in high dimensions. More importantly, operations required for reachability analysis, such as linear maps $M \otimes \mathcal{Z} := \{Mz \mid z \in \mathcal{Z}\}$ ($M \in \mathbb{R}^{q \times n}$) and Minkowski addition $\mathcal{Z}_1 \oplus \mathcal{Z}_2 := \{z_1 + z_2 \mid z_1 \in \mathcal{Z}_1, z_2 \in \mathcal{Z}_2\}$ can be computed efficiently and exactly, and others such as convex hull computation can be tightly over-approximated [8].

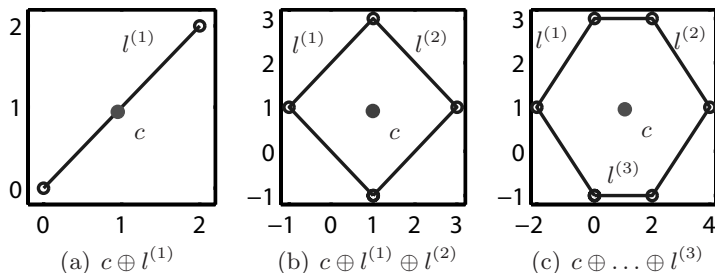


Figure 1: Step-by-step construction of a zonotope.

```

1 Z1 = zonotope([1 1 1; 1 -1 1]); % create zonotope Z1
2 Z2 = zonotope([-1 1 0; 1 0 1]); % create zonotope Z2
3 A = [0.5 1; 1 0.5]; % numerical matrix A
4
5 %% set operations
6 Z3 = Z1 + Z2; % Minkowski addition
7 Z4 = A*Z3; % linear map
8
9 figure; hold on
10 plot(Z1, [1 2], 'b'); % plot Z1 in blue
11 plot(Z2, [1 2], 'g'); % plot Z2 in green
12 plot(Z3, [1 2], 'r'); % plot Z3 in red
13 plot(Z4, [1 2], 'k'); % plot Z4 in black
14
15 %% conversion to other set representations
16 P = polytope(Z4) % convert to and display halfspace representation
17 I = interval(Z4) % convert to and display interval
18
19 figure; hold on
20 plot(Z4); % plot Z4
21 plot(I, [1 2], 'g'); % plot intervalhull in green

```

2.3 Zonotope Bundles

A disadvantage of zonotopes is that they are not closed under intersection, i.e., the intersection of two zonotopes does not return a zonotope in general. In order to overcome this disadvantage, zonotope bundles are introduced in [6]. Given a finite set of zonotopes \mathcal{Z}_i , a zonotope bundle is $\mathcal{Z}^\cap = \bigcap_{i=1}^s \mathcal{Z}_i$, i.e., the intersection of zonotopes \mathcal{Z}_i . Note that the intersection is not computed, but the zonotopes \mathcal{Z}_i are stored in a list, which we write as $\mathcal{Z}^\cap = \{\mathcal{Z}_1, \dots, \mathcal{Z}_s\}^\cap$.

```
1 Z{1} = zonotope([1 1 1; 1 -1 1]); % create zonotope Z1;
2 Z{2} = zonotope([-1 1 0; 1 0 1]); % create zonotope Z2;
3 Zb = zonotopeBundle(Z); % instantiate zonotope bundle from Z1, Z2
4 vol = volume(Zb) % compute and display volume of zonotope bundle
5
6 figure; hold on
7 plot(Z{1}); % plot Z1
8 plot(Z{2}); % plot Z2
9 plotFilled(Zb, [1 2], [.675 .675 .675], 'EdgeColor', 'none'); % plot Zb in gray
10
11 % zonotope bundles are closed under intersections, but Minkowski addition
12 % is no longer exact
13
14 %% Example 1
15 % convert to polytopes
16 Pb = polytope(Zb);
17 P1 = polytope(Z{1});
18
19 % compute exact Minkowski addition
20 Pres = Pb + P1;
21
22 % compute over-approximate addition
23 Zres = Zb + Z{1};
24
25 % compare results
26 figure; hold on
27 plot(Pres); % plot Pres
28 plotFilled(Zres, [1 2], [.675 .675 .675], 'EdgeColor', 'none'); % plot Zres
29
30 %% Example 2
31 % generate new zonotopes
32 Z{1} = zonotope([0 0.5 1 2; 0 0.5 2 1]);
33 Z{2} = zonotope([0 3 -0.7; 0 3 0.7]);
34
35 Zadd = zonotope([0 2 0; 0 0 1]); % create zonotope to be added;
36 Padd = polytope(Zadd); % convert zonotope to polytope
37
38 Zbundle = zonotopeBundle(Z); % create zonotope bundle
39 Pmpt = polytope(Zbundle); % convert zonotope to polytope
40
41 P_final = Pmpt + Padd; % compute exact Minkowski addition for polytopes
42 Zbundle_final = Zbundle + Zadd; % compute overapprox. Minkowski addition
43
44 %plot results
45 figure; hold on
46 plot(Zbundle_final); % plot Zbundle_final
47 plotFilled(P_final, [1 2], [.675 .675 .675], 'EdgeColor', 'none'); % plot P_final
```

2.4 Constrained Zonotopes

An extension of zonotopes described in Sec. 2.2 are constrained zonotopes, which are introduced in [11]. A constrained zonotope is defined as a zonotope with additional equality constraints on the factors β_i :

$$\mathcal{Z}_c = \left\{ c + G\beta \mid \|\beta\|_\infty \leq 1, A\beta = b \right\}, \quad (2)$$

where $c \in \mathbb{R}^n$ is the zonotope center, $G \in \mathbb{R}^{n \times p}$ is the zonotope generator matrix and $\beta \in \mathbb{R}^p$ is the vector of zonotope factors. The equality constraints are parametrized by the matrix $A \in \mathbb{R}^{q \times p}$ and the vector $b \in \mathbb{R}^q$. Constrained zonotopes are able to describe arbitrary polytopes, and are therefore a more general set representation than zonotopes. The main advantage compared to a polytope representation using inequality constraints is that constrained zonotopes inherit the excellent scaling properties of zonotopes for increasing state space dimensions, since constrained zonotopes are also based on a generator representation for sets.

```

1 Z{1} = zonotope([1 1 1; 1 -1 1]); % create zonotope Z1;
2 Z{2} = zonotope([-1 1 0; 1 0 1]); % create zonotope Z2;
3 Zb = zonotopeBundle(Z); % instantiate zonotope bundle from Z1, Z2
4 Zc = conZonotope(Zb); % convert to constrained zonotope
5
6 figure; hold on
7 plot(Z{1}); % plot Z1
8 plot(Z{2}); % plot Z2
9 plotFilled(Zc, [1 2], [.675 .675 .675], 'EdgeColor', 'none'); % plot Zb in gray
10
11 % constrained zonotopes are closed under intersection and Minkowski addition
12
13 %% Example 1
14 % convert to polytopes
15 Pc = mptPolytope(Zc);
16 P1 = polytope(Z{1});
17
18 % compute Minkowski addition for polytopes
19 Pres = Pc + P1;
20
21 % compute Minkowski addition for constrained zonotopes
22 Zres = Zc + Z{1};
23
24 % compare results
25 figure; hold on
26 plot(Pres); % plot P
27 plotFilled(Zres, [1 2], [.675 .675 .675], 'EdgeColor', 'none'); % plot Zres
28
29 %% Example 2
30 % generate new zonotopes
31 Z{1} = zonotope([0 0.5 1 2; 0 0.5 2 1]);
32 Z{2} = zonotope([0 3 -0.7; 0 3 0.7]);
33
34 Zadd = zonotope([0 2 0; 0 0 1]); % create zonotope to be added;
35 Padd = polytope(Zadd); % convert zonotope to polytope
36
37 Zb = zonotopeBundle(Z); % create zonotope bundle
38 Zc = conZonotope(Zb); % convert to constrained zonotope
39 P = mptPolytope(Zc); % convert zonotope to polytope
40
41 P_final = P + Padd; % compute Minkowski addition for polytopes
42 Zc_final = Zc + Zadd; % compute Minkowski addition for zonotopes

```

```

43
44 %plot results
45 figure; hold on
46 plot(Zc_final); % plot Zbundle_final
47 plotFilled(P_final, [1 2], [.675 .675 .675], 'EdgeColor', 'none'); % plot P_final

```

2.5 Polynomial Zonotopes

Zonotopes are a very efficient representation for reachability analysis of linear systems [8] and of nonlinear systems that can be well abstracted by linear differential inclusions [1]. However, more advanced techniques, such as in [2], abstract more accurately to nonlinear difference inclusions. As a consequence, linear maps of reachable sets are replaced by nonlinear maps. Zonotopes are not closed under nonlinear maps and are not particularly good at over-approximating them. For this reason, polynomial zonotopes are introduced in [2]. Polynomial zonotopes are a new non-convex set representation and can be efficiently stored and manipulated. The new representation shares many similarities with Taylor models [9] (as briefly discussed later) and is a generalization of zonotopes. Please note that a zonotope cannot be represented by a Taylor model.

Given a *starting point* $c \in \mathbb{R}^n$, multi-indexed *generators* $f^{([i],j,k,\dots,m)} \in \mathbb{R}^n$, and single-indexed *generators* $g^{(i)} \in \mathbb{R}^n$, a polynomial zonotope is defined as

$$\begin{aligned}
\mathcal{PZ} = \left\{ c + \sum_{j=1}^p \beta_j f^{([1],j)} + \sum_{j=1}^p \sum_{k=j}^p \beta_j \beta_k f^{([2],j,k)} + \dots + \sum_{j=1}^p \sum_{k=j}^p \dots \sum_{m=l}^p \underbrace{\beta_j \beta_k \dots \beta_m}_{\eta \text{ factors}} f^{([\eta],j,k,\dots,m)} \right. \\
\left. + \sum_{i=1}^q \gamma_i g^{(i)} \Big| \beta_i, \gamma_i \in [-1, 1] \right\}. \quad (3)
\end{aligned}$$

The scalars β_i are called *dependent factors*, since changing their values does not only affect the multiplication with one generator, but with other generators too. On the other hand, the scalars γ_i only affect the multiplication with one generator, so they are called *independent factors*. The number of dependent factors is p , the number of independent factors is q , and the polynomial order η is the maximum power of the scalar factors β_i . The order of a polynomial zonotope is defined as the number of generators ξ divided by the dimension, which is $\rho = \frac{\xi}{n}$. For a concise notation and later derivations, we introduce the matrices

$$\begin{aligned}
E^{[i]} &= [\underbrace{f^{([i],1,1,\dots,1)}}_{=:e^{([i],1)}} \dots \underbrace{f^{([i],p,p,\dots,p)}}_{=:e^{([i],p)}}] \text{ (all indices are the same value),} \\
F^{[i]} &= [f^{([i],1,1,\dots,1,2)} \ f^{([i],1,1,\dots,1,3)} \ \dots \ f^{([i],1,1,\dots,1,p)} \\
&\quad f^{([i],1,1,\dots,2,2)} \ f^{([i],1,1,\dots,2,3)} \ \dots \ f^{([i],1,1,\dots,2,p)} \\
&\quad f^{([i],1,1,\dots,3,3)} \ \dots] \text{ (not all indices are the same value),} \\
G &= [g^{(1)} \ \dots \ g^{(q)}],
\end{aligned}$$

and $E = [E^{[1]} \ \dots \ E^{[\eta]}]$, $F = [F^{[2]} \ \dots \ F^{[\eta]}]$ ($F^{[i]}$ is only defined for $i \geq 2$). Note that the indices in $F^{[i]}$ are ascending due to the nested summations in (3). In short form, a polynomial zonotope is written as $\mathcal{PZ} = (c, E, F, G)$.

For a given polynomial order i , the total number of generators in $E^{[i]}$ and $F^{[i]}$ is derived using the number $\binom{p+i-1}{i}$ of combinations of the scalar factors β with replacement (i.e., the same factor can be used again). Adding the numbers for all polynomial orders and adding the number of independent generators q , results in $\xi = \sum_{i=1}^{\eta} \binom{p+i-1}{i} + q$ generators, which is in $\mathcal{O}(p^\eta)$ with

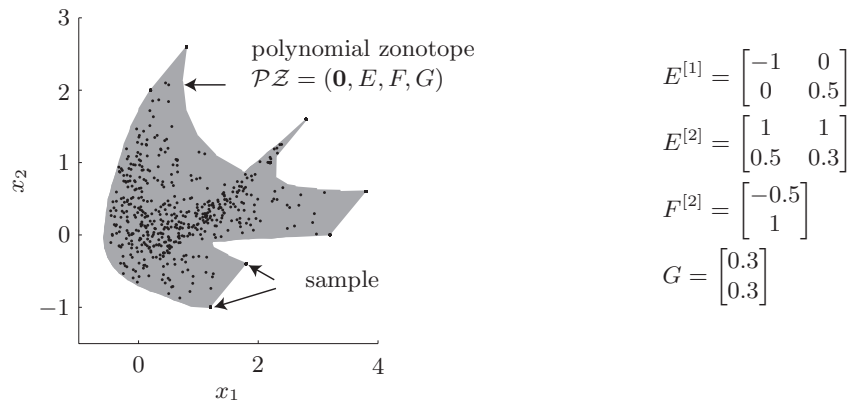


Figure 2: Over-approximative plot of a polynomial zonotope as specified in the figure. Random samples of possible values demonstrate the accuracy of the over-approximative plot.

respect to p . The non-convex shape of a polynomial zonotope with polynomial order 2 is shown in Fig. 2.

```

1 c = [0;0]; % starting point
2 E1 = diag([-1,0.5]); % generators of factors with identical indices
3 E2 = [1 1; 0.5 0.3]; % generators of factors with identical indices
4 F = [-0.5; 1]; % generators of factors with different indices
5 G = [0.3; 0.3]; % independent generators
6
7 qZ = quadZonotope(c,E1,E2,F,G); % instantiate quadratic zonotope
8 Z = zonotope(qZ) % over-approximate by a zonotope
9
10 figure; hold on
11 plot(Z); % plot Z
12 plotFilled(qZ,[1 2],7,[],[.6 .6 .6],'EdgeColor','none'); % plot qZ

```

2.6 Further Set Representations

These set representations are also implemented in CORA, but not covered in this tutorial:

- affine representations
- capsules
- ellipsoids
- halfspaces
- polytopes
- probabilistic zonotopes
- Taylor models
- zoo (mixture of presentations)

3 Linear Discrete-Time Systems

One of the simplest problems in reachability analysis is to compute the reachable set of a linear discrete time system

$$x(t_{k+1}) = Ax(t_k) + u(t_k),$$

where $x(0) \in \mathcal{X}_0 \subset \mathbb{R}^n$, $u(t_k) \in \mathcal{U} \subset \mathbb{R}^m$, and $A \in \mathbb{R}^{n \times n}$. The reachable set $\mathcal{R}(t_k)$ can be trivially computed as

$$\mathcal{R}(t_{k+1}) = A\mathcal{R}(t_k) + u(t_k),$$

where $\mathcal{R}(0) = \mathcal{X}_0$. Since the representation size of the reachable set grows with each iteration, we additionally have to reduce the size of the reachable set in each iteration.

```
1 function Rcont = tutorial_reachLinDiscrete(A,B,options)
2
3 % Input to be added
4 Uadd = B*options.U;
5
6 % Initialize reachable set
7 Rnext = options.R0;
8
9 %time period
10 tVec = options.tStart:options.timeStep:options.tFinal;
11
12 % init Rcont
13 Rcont = cell(length(tVec)-1,1);
14
15 % loop over all reachability steps
16 for i = 2:length(tVec)
17
18     % order reduction
19     Rcont{i-1} = reduce(Rnext,options.reductionTechnique,options.zonotopeOrder);
20
21     %compute next reachable set
22     Rnext = A*Rnext + Uadd;
23 end
```

It remains to specify the system and plot the results

```
1 % system matrix
2 A = [0.95 -0.15 0 0 0; ...
3       0.15 0.95 0 0 0; ...
4       0 0 0.9 0.05 0; ...
5       0 0 -0.05 0.9 0; ...
6       0 0 0 0 0.92];
7
8 % input matrix
9 B = 1;
10
11 %% set reachability options
12 options.tStart=0; %start time
13 options.tFinal=5; %final time
14 options.timeStep=0.04; %time step size for reachable set computation
15 options.R0=zonotope([ones(length(A),1),0.1*eye(length(A))]); %initial set
16 options.reductionTechnique='girard'; %technique for order reduction
17 options.zonotopeOrder=200; %zonotope order
18 options.U=zonotope([zeros(5,1),0.02*diag([0.1, 0.3, 0.1, 0.3, 0.3])]); %input
19
```

```
20 % compute reachable set
21 Rcont = tutorial_reachLinDiscrete(A,B,options);
22
23 %% plot results
24 for plotRun=1:2
25     % plot different projections
26     if plotRun==1
27         projectedDimensions=[1 2];
28     elseif plotRun==2
29         projectedDimensions=[3 4];
30     end
31
32     figure;
33     hold on
34
35     %plot reachable sets
36     for i=1:length(Rcont)
37         plotFilled(Rcont{i},projectedDimensions,[.8 .8 .8],'EdgeColor','none');
38     end
39
40     %plot initial set
41     plot(options.R0,projectedDimensions,'b-','lineWidth',2);
42
43     %label plot
44     xlabel(['x_{',num2str(projectedDimensions(1)),'}']);
45     ylabel(['x_{',num2str(projectedDimensions(2)),'}']);
46 end
47
```

4 Linear Continuous-Time Systems

While reachable sets can be easily computed for linear discrete time systems, one requires more advanced algorithms for linear continuous time systems

$$\dot{x}(t) = Ax(t) + u(t), \quad (4)$$

where $x(0) \in \mathcal{X}_0 \subset \mathbb{R}^n$, $u(t) \in \mathcal{U} \subset \mathbb{R}^m$, and $A \in \mathbb{R}^{n \times n}$. Since one can apply the superposition principle to linear systems, we first compute the homogeneous solution ($u(t) = 0$) followed by the particular solution ($x(0) = 0$).

4.1 Homogeneous Solution for Points in Time

It is well known that the homogeneous solution of (4) is

$$x(t) = e^{At}x(0),$$

where

$$\begin{aligned}
 e^{At} &= \sum_{i=0}^{\infty} \frac{(At)^i}{i!} \\
 &= I + \frac{At}{1!} + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \dots \\
 \frac{d}{dt}e^{At} &= \mathbf{0} + A + 2\frac{At}{2!}A + 3\frac{(At)^2}{3!}A + \dots \\
 &= A\left(I + \frac{At}{1!} + \frac{(At)^2}{2!} + \dots\right) = Ae^{At}
 \end{aligned}$$

Test of solution:

$$\begin{aligned}
 x(t) &= e^{At}x(0) \\
 \frac{d}{dt}x(t) &= \frac{d}{dt}e^{At}x(0) = A \underbrace{e^{At}x(0)}_{x(t)} \quad \checkmark
 \end{aligned}$$

How to compute the infinite series?

Answer: make it finite

$$\begin{aligned}
 e^{At} &= \sum_{i=0}^l \frac{(At)^i}{i!} + \underbrace{\sum_{i=l+1}^{\infty} \frac{(At)^i}{i!}}_{=:E} \\
 \|E\| &\leq \sum_{i=l+1}^{\infty} \frac{(\|A\|t)^i}{i!}
 \end{aligned}$$

Ratio of terms:

$$\begin{aligned}
 \frac{\text{"2nd term"}}{\text{"1st term"}} &= \frac{(\|A\|t)^{l+2}}{(l+2)!} \frac{(l+1)!}{(\|A\|t)^{l+1}} = \frac{\|A\|t}{l+2} =: \omega \\
 \frac{\text{"3rd term"}}{\text{"2nd term"}} &= \frac{\|A\|t}{l+3} < \omega
 \end{aligned}$$

Assume ω as worst-case ratio for all terms:

$$\begin{aligned}
 \|E\| &\leq \sum_{i=l+1}^{\infty} \frac{(\|A\|t)^i}{i!} \leq \frac{(\|A\|t)^{l+1}}{(l+1)!} \underbrace{(1 + \omega + \omega^2 + \omega^3 + \dots)}_{= \frac{1}{1-\omega} \text{ for } \omega < 1} \\
 \implies e^{At} &\in \sum_{i=0}^l \frac{(At)^i}{i!} + \begin{bmatrix} \underbrace{[-1, 1]} \\ [-1, 1] & \dots \\ \vdots & & [-1, 1] \end{bmatrix} \frac{(\|A\|t)^{l+1}}{(l+1)!} \frac{1}{1-\omega}
 \end{aligned}$$

4.2 Homogeneous Solution for Time Intervals

$$\tilde{x}(t) = x(0) + \frac{t}{r}(x(r) - x(0)) \text{ for } t \in [0, r]$$

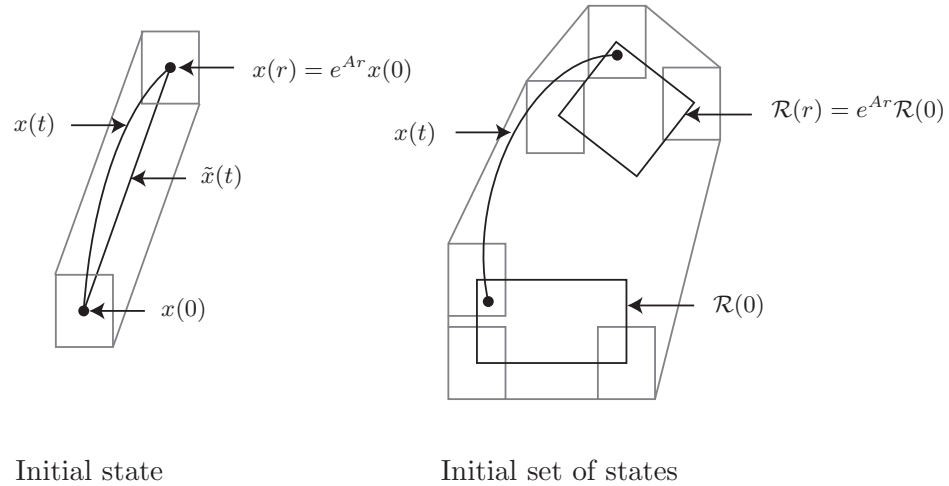


Figure 3: Enclosure of a trajectory for an initial state and a set of initial states.

$$\begin{aligned}
 x(t) - \tilde{x}(t) &= x(t) - x(0) - \frac{t}{r}(x(r) - x(0)) \\
 &= e^{At}x(0) - x(0) - \frac{t}{r}(e^{Ar}x(0) - x(0)) \\
 &= \underbrace{\left[e^{At} - I - \frac{t}{r}(e^{Ar} - I) \right]}_{\subseteq F} x(0) \\
 &\implies x(t) \in \tilde{x}(t) + Fx(0)
 \end{aligned}$$

Insert Taylor series:

$$\begin{aligned}
 F &= I + At + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \dots + E(t) - I - \frac{t}{r} \left(I + Ar + \frac{(Ar)^2}{2!} + \frac{(Ar)^3}{3!} + \dots + E(r) - I \right) \\
 &= At + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \dots + E(t) - At - \frac{A^2rt}{2!} + \frac{A^3r^2t}{3!} - \dots - \frac{t}{r}E(r) \\
 &= \frac{A^2(t^2 - tr)}{2!} + \frac{A^3(t^3 - tr^2)}{3!} + \dots + E(t) - \frac{t}{r}E(r) \\
 &= \sum_{i=2}^l \frac{A^i(t^i - tr^{i-1})}{i!} + E(t) - \frac{t}{r}E(r)
 \end{aligned}$$

What is the bound for $\gamma(t) := t^i - tr^{i-1}$ for $t \in [0, r]$?

$$\begin{aligned}
 \frac{d}{dt}(t^i - tr^{i-1}) &= it^{i-1} - r^{i-1} = 0 \\
 t^{i-1} &= \frac{1}{i}r^{i-1} \\
 t &= i^{\frac{-1}{i-1}}r \\
 \frac{d^2}{dt^2}\gamma(t) &= \frac{d}{dt}it^{i-1} - r^{i-1} = i(i-1)t^{i-2} > 0
 \end{aligned}$$

Thus, $\gamma(t)$ is a convex function and thus we obtain the minimum from above as $t_{min} = i^{\frac{-1}{i-1}}r$.

Inserting t_{min} results in

$$\gamma(t_{min}) = i^{-\frac{i}{i-1}} r^i - i^{-\frac{1}{i-1}} r r^{i-1} = \underbrace{\left(i^{-\frac{i}{i-1}} - i^{-\frac{1}{i-1}} \right)}_{\substack{1 \\ i^{\frac{i}{i-1}} - \frac{1}{i^{\frac{1}{i-1}}} \\ < 0}} \underbrace{r^i}_{> 0} < 0$$

Since the function is concave, the maximum is to be found at the boundaries ($t = 0, t = r$):

$$\begin{aligned} \gamma(0) &= 0, \quad \gamma(r) = r^i - r r^{i-1} = 0 \\ \implies \forall t \in [0, r] : \gamma(t) &\in [i^{-\frac{i}{i-1}} - i^{-\frac{1}{i-1}}, 0] r^i \end{aligned}$$

It remains to bound $E(t) - \frac{t}{r} E(r)$ for $t \in [0, r]$, where $E(t) \in [-\mathbf{1}, \mathbf{1}]^{n \times n} \phi(t)$.

$$\begin{aligned} \phi(t) \text{ is monotonically increasing} &\implies \forall t \in [0, r] : \phi(t) \in [0, 1] \phi(r) \\ &\implies \forall t \in [0, r] : \phi(t) - \frac{t}{r} \phi(r) \in [0, 1] \phi(r) - [0, 1] \phi(r) = [-1, 1] \phi(r) \\ &\implies \forall t \in [0, r] : E(t) - \frac{t}{r} E(r) \in [-\mathbf{1}, \mathbf{1}] \phi(r) \end{aligned}$$

Final result:

$$F = \sum_{i=2}^l [i^{-\frac{i}{i-1}} - i^{-\frac{1}{i-1}}, 0] r^i \frac{A^i}{i!} + [-\mathbf{1}, \mathbf{1}] \frac{(\|A\|r)^{l+1}}{(l+1)!} \frac{1}{1-\omega}$$

4.3 Input Solution

The derivation of the input solution is more complicated so that we just provide the result:

$$\mathcal{R}^p(r) = \sum_{i=0}^l \left(\frac{A^i r^{i+1}}{(i+1)!} \mathcal{U} \right) + E(r) r \mathcal{U}$$

4.4 Realization in CORA

We specify the system similarly to the one for discrete time:

```

1 %% set reachability options
2 options.tStart=0; %start time
3 options.tFinal=5; %final time
4 options.R0=zonotope([ones(5,1), 0.1*eye(5)]); %initial set
5
6 options.timeStep=0.04; %time step size for reachable set computation
7 options.taylorTerms=4; %number of taylor terms for reachable sets
8 options.zonotopeOrder=200; %zonotope order
9 options.originContained=0; %specify whether input set contains origin
10 options.reductionTechnique='girard'; %specify order reduction method
11
12 uTrans=[1; 0; 0; 0.5; -0.5];
13 options.uTrans=uTrans; %center of input set
14 options.U=0.5*zonotope([zeros(5,1), diag([0.2, 0.5, 0.2, 0.5, 0.5])]); %input set
15
16 %% set simulation options

```

```
17 simOptions.tStart=0; %start time
18 simOptions.tFinal=5; %final time
19 simOptions.R0=zonotope([ones(5,1),0.1*eye(5)]);
20
21 simOptions.uTrans=[1; 0; 0; 0.5; -0.5]; %center of input set
22 simOptions.U=0.5*zonotope([zeros(5,1),diag([0.2, 0.5, 0.2, 0.5, 0.5])]);
23
24 %% specify continuous dynamics
25 A=[-1 -4 0 0 0; 4 -1 0 0 0; 0 0 -3 1 0; 0 0 -1 -3 0; 0 0 0 0 -2];
26 B=1;
27 fiveDimSys=linearSys('fiveDimSys',A,B); %initialize system
28
29
30 %% compute reachable set using zonotopes
31 tic
32 Rcont = reach(fiveDimSys, options);
33 tComp = toc;
34 disp(['computation time of reachable set: ',num2str(tComp)]);
35
36 %% create random simulations
37 runs = 60;
38 fractionVertices = 0.5;
39 fractionInputVertices = 0.5;
40 inputChanges = 6;
41 simRes = simulate_random(fiveDimSys, simOptions, runs, fractionVertices, ...
42     fractionInputVertices, inputChanges);
43
44 %% plot results
45 for plotRun=1:2
46     % plot different projections
47     if plotRun==1
48         projDim=[1 2];
49     elseif plotRun==2
50         projDim=[3 4];
51     end
52
53     figure;
54     hold on
55
56     %plot reachable sets
57     for i=1:length(Rcont)
58         plotFilled(Rcont{i},projDim,[.8 .8 .8],'EdgeColor','none');
59     end
60
61     %plot initial set
62     plot(options.R0,projDim,'w-','lineWidth',2);
63
64     %plot simulation results
65     for i=1:length(simRes.t)
66         plot(simRes.x{i}(:,projDim(1)),simRes.x{i}(:,projDim(2)),'Color',[0 0 0]);
67     end
68
69     %label plot
70     xlabel(['x_',num2str(projDim(1)),'']);
71     ylabel(['x_',num2str(projDim(2)),'']);
72 end
```

5 Loading Models from SpaceEx

To load SpaceEx models (stored as XML files) into CORA, one only has to execute a simple command:

```
spaceex2cora('model.xml');
```

This command creates a CORA model in `/models/SpaceExConverted` under a folder with the identical name as the SpaceExModel. If the SpaceEx model contains nonlinear differential equations, additional dynamics files are stored in the same folder. Below we present as an example the converted model of the bouncing ball model from SpaceEx:

```
1 function HA = bball(~)
2
3
4 %% Generated on 07-Aug-2018
5
6 %-----Automaton created from Component 'system'-----
7
8 %% Interface Specification:
9 % This section clarifies the meaning of state & input dimensions
10 % by showing their mapping to SpaceEx variable names.
11
12 % Component 1 (system.ball):
13 % state x := [x; v]
14 % input u := [uDummy]
15
16 %-----Component system.ball-----
17
18 %-----State always-----
19
20 %% equation:
21 % x' == v & v' == -g
22 dynA = ...
23 [0,1;0,0];
24 dynB = ...
25 [0;0];
26 dync = ...
27 [0;-1];
28 dynamics = linearSys('linearSys', dynA, dynB, dync);
29
30 %% equation:
31 % x >= 0
32 invA = ...
33 [-1,0];
34 invb = ...
35 [-0];
36 invOpt = struct('A', invA, 'b', invb);
37 inv = mptPolytope(invOpt);
38
39 trans = {};
40 %% equation:
41 % v' := -c*v
42 resetA = ...
43 [1,0;0,-0.75];
44 resetb = ...
45 [0;0];
```

```
46 reset = struct('A', resetA, 'b', resetb);
47
48 %% equation:
49 %    x <= eps & v < 0
50 guardA = ...
51 [1,0;0,1];
52 guardb = ...
53 [-0;-0];
54 guardOpt = struct('A', guardA, 'b', guardb);
55 guard = mptPolytope(guardOpt);
56
57 trans{1} = transition(guard, reset, 1, 'dummy', 'names');
58
59 loc{1} = location('S1',1, inv, trans, dynamics);
60
61
62
63 HA = hybridAutomaton(loc);
64
65
66 end
```

At the beginning of each automatically created model, we list the state and inputs so that the created models can be interpreted more easily using the variable names from the SpaceEx model. These variable names are later replaced by the state vector x and the input vector u to make use of matrix multiplications in MATLAB for improved efficiency. Next, the dynamic equations, guard sets, invariants, transitions, and locations are created.

How to obtain SpaceEx models?

- Use the SpaceEx model editor: spaceex.imag.fr/download-6
- Convert Simulink models to SpaceEx: github.com/nikos-kekatos/SL2SX

6 Nonlinear Systems

So far, reachable sets of linear continuous systems have been presented. Although a fairly large group of dynamic systems can be described by linear continuous systems, the extension to nonlinear continuous systems is an important step for the analysis of more complex systems. The analysis of nonlinear systems is much more complicated since many valuable properties are no longer valid. One of them is the superposition principle, which allows the homogeneous and the inhomogeneous solution to be obtained separately. Another is that reachable sets of linear systems can be computed by a linear map. This makes it possible to exploit that geometric representations such as ellipsoids, zonotopes, and polytopes are closed under linear transformations, i.e., they are again mapped to ellipsoids, zonotopes and polytopes, respectively. In CORA, reachability analysis of nonlinear systems is based on abstraction. We present abstraction by linear systems as presented in [1, Section 3.4] and by polynomial systems as presented in [2]. Since the abstraction causes additional errors, the abstraction errors are determined in an over-approximative way and added as an additional uncertain input so that an over-approximative computation is ensured.

6.1 Main Principle

General nonlinear continuous systems with uncertain parameters and Lipschitz continuity are considered. In analogy to the previous linear systems, the initial state $x(0)$ can take values from a set $\mathcal{X}_0 \subset \mathbb{R}^n$ and the input u takes values from a set $\mathcal{U} \subset \mathbb{R}^m$. The evolution of the state x is defined by the following differential equation:

$$\dot{x}(t) = f(x(t), u(t)), \quad x(0) \in \mathcal{X}_0 \subset \mathbb{R}^n, \quad u(t) \in \mathcal{U} \subset \mathbb{R}^m,$$

where $u(t)$ and $f(x(t), u(t))$ are assumed to be globally Lipschitz continuous so that the Taylor expansion for the state and the input can always be computed, a condition required for the abstraction.

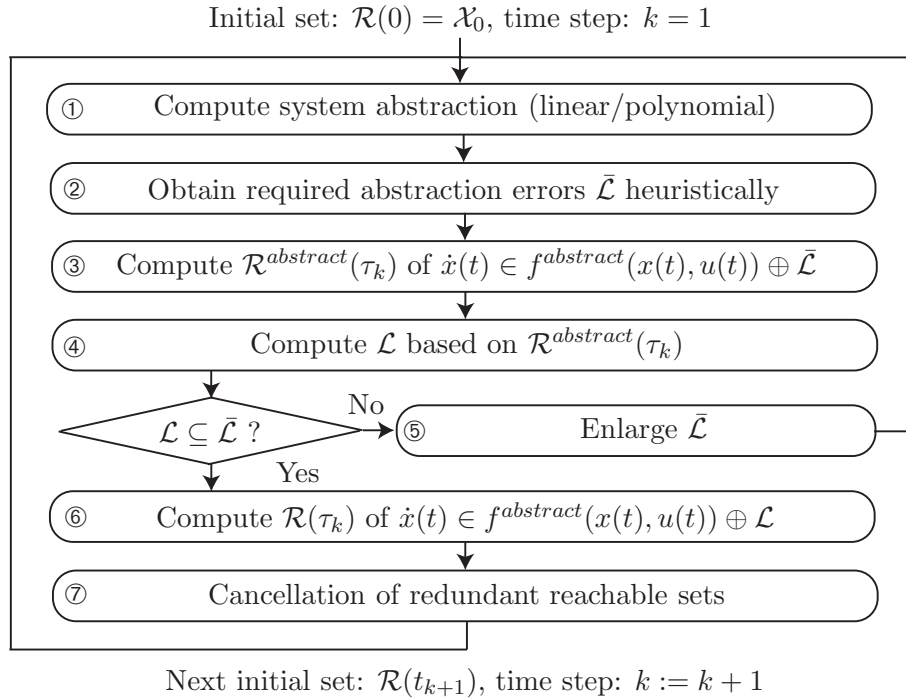


Figure 4: Computation of reachable sets – overview.

A brief visualization of the overall concept for computing the reachable set is shown in Fig. 4. As in the previous approaches, the reachable set is computed iteratively for time intervals $t \in \tau_k = [kr, (k+1)r]$ where $k \in \mathbb{N}^+$. The procedure for computing the reachable sets of the consecutive time intervals is as follows:

- ① The nonlinear system $\dot{x}(t) = f(x(t), u(t))$ is either abstracted to a linear system, or after introducing $z = [x^T, u^T]^T$, to a polynomial system of the form

$$\begin{aligned} \dot{x}_i = f^{abstract}(x, u) = & w_i + \frac{1}{1!} \sum_{j=1}^o C_{ij} z_j(t) + \frac{1}{2!} \sum_{j=1}^o \sum_{k=1}^o D_{ijk} z_j(t) z_k(t) \\ & + \frac{1}{3!} \sum_{j=1}^o \sum_{k=1}^o \sum_{l=1}^o E_{ijkl} z_j(t) z_k(t) z_l(t) + \dots \end{aligned} \quad (5)$$

The set of abstraction errors \mathcal{L} ensures that $f(x, u) \in f^{abstract}(x, u) \oplus \mathcal{L}$, which allows the reachable set to be computed in an over-approximative way.

- ② Next, the set of required abstraction errors $\bar{\mathcal{L}}$ is obtained heuristically.

- ③ The reachable set $\mathcal{R}^{abstract}(\tau_k)$ of $\dot{x}(t) \in f^{abstract}(x(t), u(t)) \oplus \bar{\mathcal{L}}$ is computed.
- ④ The set of abstraction errors \mathcal{L} is computed based on the reachable set $\mathcal{R}^{abstract}(\tau_k)$.
- ⑤ When $\mathcal{L} \not\subseteq \bar{\mathcal{L}}$, the abstraction error is not admissible, requiring the assumption $\bar{\mathcal{L}}$ to be enlarged. If several enlargements are not successful, one has to split the reachable set and continue with one more partial reachable set from then on.
- ⑥ If $\mathcal{L} \subseteq \bar{\mathcal{L}}$, the abstraction error is accepted and the reachable set is obtained by using the tighter abstraction error: $\dot{x}(t) \in f^{abstract}(x(t), u(t)) \oplus \mathcal{L}$.
- ⑦ It remains to increase the time step ($k := k + 1$) and cancel redundant reachable sets that are already covered by previously computed reachable sets. This decreases the number of reachable sets that have to be considered in the next time interval.

The necessity of splitting reachable sets is indicated in the workspace outputs using the keyword `split`.

6.2 Quadrotor Example

We study the dynamics of a quadrotor as derived in [7, eq. (16) - (19)]. Let us first introduce the variables required to describe the model: The inertial (north) position x_1 , the inertial (east) position x_2 , the altitude x_3 , the longitudinal velocity x_4 , the lateral velocity x_5 , the vertical velocity x_6 , the roll angle x_7 , the pitch angle x_8 , the yaw angle x_9 , the roll rate x_{10} , the pitch rate x_{11} , and the yaw rate x_{12} . We further require the following parameters: gravity constant $g = 9.81$ [m/s²], radius of center mass $R = 0.1$ [m], distance of motors to center mass $l = 0.5$ [m], motor mass $M_{rotor} = 0.1$ [kg], center mass $M = 1$ [kg], and total mass $m = M + 4M_{rotor}$.

From the above parameters we can compute the moments of inertia as

$$\begin{aligned} J_x &= \frac{2}{5} M R^2 + 2l^2 M_{rotor}, \\ J_y &= J_x, \\ J_z &= \frac{2}{5} M R^2 + 4l^2 M_{rotor}. \end{aligned}$$

Finally, we can write the set of ordinary differential equations for the quadrotor according to [7, eq. (16) - (19)]:

$$\left\{ \begin{array}{l} \dot{x}_1 = \cos(x_8) \cos(x_9) x_4 + \left(\sin(x_7) \sin(x_8) \cos(x_9) - \cos(x_7) \sin(x_9) \right) x_5 \\ \quad + \left(\cos(x_7) \sin(x_8) \cos(x_9) + \sin(x_7) \sin(x_9) \right) x_6 \\ \dot{x}_2 = \cos(x_8) \sin(x_9) x_4 + \left(\sin(x_7) \sin(x_8) \sin(x_9) + \cos(x_7) \cos(x_9) \right) x_5 \\ \quad + \left(\cos(x_7) \sin(x_8) \sin(x_9) - \sin(x_7) \cos(x_9) \right) x_6 \\ \dot{x}_3 = \sin(x_8) x_4 - \sin(x_7) \cos(x_8) x_5 - \cos(x_7) \cos(x_8) x_6 \\ \dot{x}_4 = x_{12} x_5 - x_{11} x_6 - g \sin(x_8) \\ \dot{x}_5 = x_{10} x_6 - x_{12} x_4 + g \cos(x_8) \sin(x_7) \\ \dot{x}_6 = x_{11} x_4 - x_{10} x_5 + g \cos(x_8) \cos(x_7) - \frac{F}{m} \\ \dot{x}_7 = x_{10} + \sin(x_7) \tan(x_8) x_{11} + \cos(x_7) \tan(x_8) x_{12} \\ \dot{x}_8 = \cos(x_7) x_{11} - \sin(x_7) x_{12} \\ \dot{x}_9 = \frac{\sin(x_7)}{\cos(x_8)} x_{11} + \frac{\cos(x_7)}{\cos(x_8)} x_{12} \\ \dot{x}_{10} = \frac{J_y - J_z}{J_x} x_{11} x_{12} + \frac{1}{J_x} \tau_\phi \\ \dot{x}_{11} = \frac{J_z - J_x}{J_y} x_{10} x_{12} + \frac{1}{J_y} \tau_\theta \\ \dot{x}_{12} = \frac{J_x - J_y}{J_z} x_{10} x_{11} + \frac{1}{J_z} \tau_\psi \end{array} \right.$$

To check interesting control specifications, we stabilize the quadrotor using simple PD controllers for height, roll, and pitch. The inputs to the controller are the desired values for height, roll, and pitch u_1 , u_2 , and u_3 , respectively. The equations of the controllers are

$$\begin{aligned} F &= mg - 10(x_3 - u_1) + 3x_6 && \text{(height control),} \\ \tau_\phi &= -(x_7 - u_2) - x_{10} && \text{(roll control),} \\ \tau_\theta &= -(x_8 - u_3) - x_{11} && \text{(pitch control).} \end{aligned}$$

We leave the heading uncontrolled so that we set $\tau_\psi = 0$.

The task is to change the height from 0 [m] to 1 [m] within 5 [s]. A goal region $[0.98, 1.02]$ of the height x_3 has to be reached within 5 [s] and the height has to stay below 1.4 for all times. After 1 [s] the height should stay above 0.9 [m]. The initial position of the quadrotor is uncertain in all directions within $[-0.4, 0.4]$ [m] and also the velocity is uncertain within $[-0.4, 0.4]$ [m/s] for all directions. All other values are initialized as 0.

The MATLAB code that implements the simulation and reachability analysis of the quadrotor example is:

```

1  %% set options
2  options.tStart=0; %start time
3  options.tFinal=5; %final time
4  options.x0=zeros(12,1); %initial state for simulation
5  options.R0=zonotope([options.x0,0.4*diag([ones(6,1); zeros(6,1)])]); %initial set
6  options.timeStep=0.1; %time step size for reachable set computation
7  options.taylorTerms=4; %number of taylor terms for reachable sets
8  options.zonotopeOrder=50; %zonotope order
9  options.intermediateOrder=5;
10 options.reductionTechnique='girard';
11 options.errorOrder=1;
12 options.polytopeOrder=2; %polytope order
13 options.reductionInterval=1e3;
14 options.maxError = 1*ones(12,1);
15 options.originContained = 0;
16 options.advancedLinErrorComp = 0;
17 options.tensorOrder = 2;
18 options.uTrans = [1;0;0];
19 options.U = zonotope([0;0;0]); %input for reachability analysis
20
21 %% specify continuous dynamics
22 quadcopter = nonlinearSys(12,3,@quadcopterControlledEq,options);
23
24 %% compute reachable set
25 tic
26 Rcont = reach(quadcopter, options);
27 tComp = toc;
28 disp(['computation time of reachable set: ',num2str(tComp)]);
29
30 %% create random simulations
31 runs = 60;
32 fractionVertices = 0.5;
33 fractionInputVertices = 0.5;
34 inputChanges = 6;
35 simRes = simulate_random(quadcopter, options, runs, fractionVertices, ...
36     fractionInputVertices, inputChanges);
37
38 %% plot results
39 figure;

```

```

40 hold on
41
42 %plot time elapse
43 for i=1:length(Rcont)
44     %get Uout
45     Uout1 = interval(project(Rcont{i}{1},3));
46     %obtain times
47     t1 = (i-1)*options.timeStep;
48     t2 = i*options.timeStep;
49     %generate plot areas as interval hulls
50     IH = interval([t1; infimum(Uout1)], [t2; supremum(Uout1)]);
51
52     plotFilled(IH,[1 2],[.75 .75 .75],'EdgeColor','none');
53 end
54
55 %plot simulation results
56 for i=1:(length(simRes.t))
57     plot(simRes.t{i},simRes.x{i}(:,3),'Color',0*[1 1 1]);
58 end
59
60 %label plot
61 box on
62 xlabel('t');
63 ylabel('altitude x_3');
64 axis([0,5,-0.5,1.5]);

```

The reachable set and the simulation are plotted in Fig. 5 for a time horizon of $t_f = 5$.

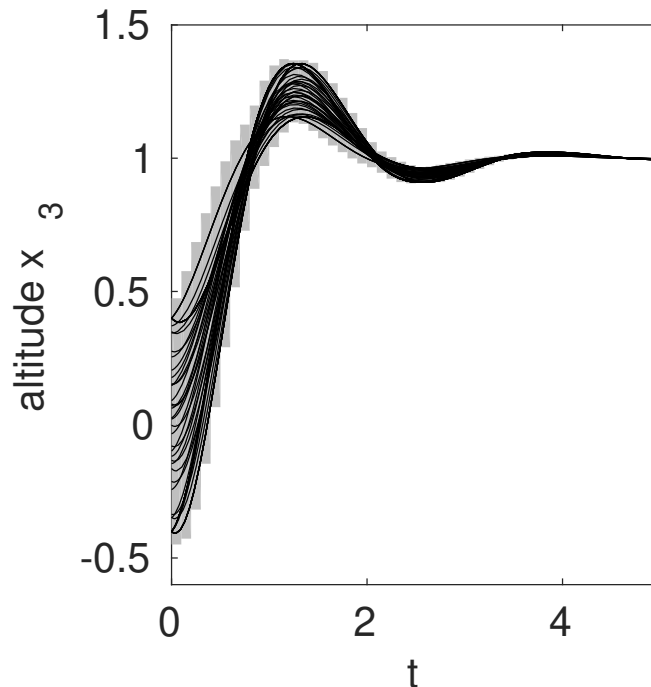


Figure 5: Illustration of the reachable set of the quadrotor. The black box shows the initial set and the black line shows the simulated trajectory.

7 Hybrid Systems

In CORA, hybrid systems are modeled by hybrid automata. Besides a continuous state x , there also exists a discrete state v for hybrid systems. The continuous initial state may take values within continuous sets while only a single initial discrete state is assumed without loss of generality³. The switching of the continuous dynamics is triggered by *guard sets*. Jumps in the continuous state are considered after the discrete state has changed. One of the most intuitive examples where jumps in the continuous state can occur is the bouncing ball example (see Sec. 7.2), where the velocity of the ball changes instantaneously when hitting the ground.

7.1 Hybrid Automata

The formal definition of the hybrid automaton is similarly defined as in [12]. The main difference is the consideration of uncertain parameters and the restrictions on jumps and guard sets. A hybrid automaton $HA = (\mathcal{V}, v^0, \mathcal{X}, \mathcal{X}^0, \mathcal{U}, \mathcal{P}, \text{inv}, \mathbf{T}, \mathbf{g}, \mathbf{h}, \mathbf{f})$, as it is considered in CORA, consists of:

- the finite set of locations $\mathcal{V} = \{v_1, \dots, v_\xi\}$ with an initial location $v^0 \in \mathcal{V}$.
- the continuous state space $\mathcal{X} \subseteq \mathbb{R}^n$ and the set of initial continuous states \mathcal{X}^0 such that $\mathcal{X}^0 \subseteq \text{inv}(v^0)$.
- the continuous input space $\mathcal{U} \subseteq \mathbb{R}^m$.
- the parameter space $\mathcal{P} \subseteq \mathbb{R}^p$.
- the mapping⁴ $\text{inv}: \mathcal{V} \rightarrow 2^{\mathcal{X}}$, which assigns an invariant $\text{inv}(v) \subseteq \mathcal{X}$ to each location v .
- the set of discrete transitions $\mathbf{T} \subseteq \mathcal{V} \times \mathcal{V}$. A transition from $v_i \in \mathcal{V}$ to $v_j \in \mathcal{V}$ is denoted by (v_i, v_j) .
- the guard function $\mathbf{g}: \mathbf{T} \rightarrow 2^{\mathcal{X}}$, which associates a guard set $\mathbf{g}((v_i, v_j))$ for each transition from v_i to v_j , where $\mathbf{g}((v_i, v_j)) \cap \text{inv}(v_i) \neq \emptyset$.
- the jump function $\mathbf{h}: \mathbf{T} \times \mathcal{X} \rightarrow \mathcal{X}$, which returns the next continuous state when a transition is taken.
- the flow function $\mathbf{f}: \mathcal{V} \times \mathcal{X} \times \mathcal{U} \times \mathcal{P} \rightarrow \mathbb{R}^n$, which defines a continuous vector field for the time derivative of x : $\dot{x} = \mathbf{f}(v, x, u, p)$.

The invariants $\text{inv}(v)$ and the guard sets $\mathbf{g}((v_i, v_j))$ are modeled by polytopes. The jump function is restricted to a linear map

$$x' = K_{(v_i, v_j)} x + l_{(v_i, v_j)}, \quad (6)$$

where x' denotes the state after the transition is taken and $K_{(v_i, v_j)} \in \mathbb{R}^{n \times n}$, $l_{(v_i, v_j)} \in \mathbb{R}^n$ are specific for a transition (v_i, v_j) . The input sets \mathcal{U}_v are modeled by zonotopes and are also dependent on the location v . Note that in order to use the results from reachability analysis of nonlinear systems, the input $u(t)$ is assumed to be locally Lipschitz continuous. The set of parameters \mathcal{P}_v can also be chosen differently for each location v .

The evolution of the hybrid automaton is described informally as follows. Starting from an initial location $v(0) = v^0$ and an initial state $x(0) \in \mathcal{X}^0$, the continuous state evolves according to the flow function that is assigned to each location v . If the continuous state is within a

³In the case of several initial discrete states, the reachability analysis can be performed for each discrete state separately.

⁴ $2^{\mathcal{X}}$ is the power set of \mathcal{X} .

guard set, the corresponding transition can be taken and has to be taken if the state would otherwise leave the invariant $\text{inv}(v)$. When the transition from the previous location v_i to the next location v_j is taken, the system state is updated according to the jump function and the continuous evolution within the next invariant.

Because the reachability of discrete states is simply a question of determining if the continuous reachable set hits certain guard sets, the focus of CORA is on the continuous reachable sets. Clearly, as for the continuous systems, the reachable set of the hybrid system has to be over-approximated in order to verify the safety of the system. An illustration of a reachable set of a hybrid automaton is given in Fig. 6.

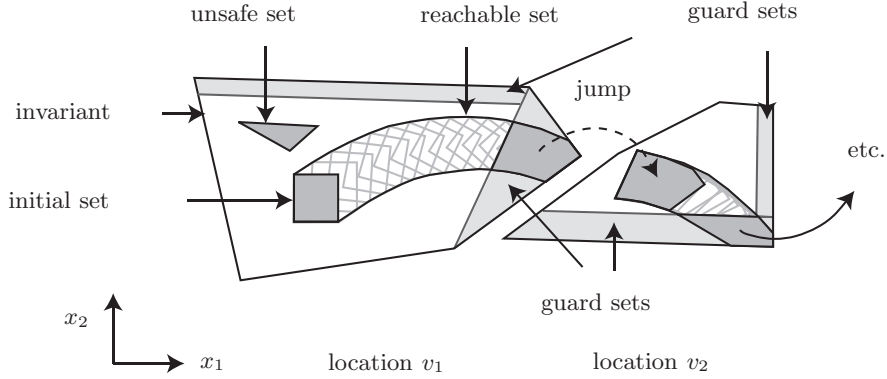


Figure 6: Illustration of the reachable set of a hybrid automaton.

7.2 Bouncing Ball Example

We demonstrate the syntax of CORA for the well-known bouncing ball example, see e.g., [13, Section 2.2.3]. Given is a ball in Fig. 7 with dynamics $\ddot{s} = -g$, where s is the vertical position and g is the gravity constant. After impact with the ground at $s = 0$, the velocity changes to $v' = -\alpha v$ ($v = \dot{s}$) with $\alpha \in [0, 1]$. The corresponding hybrid automaton can be formalized as presented in Fig. 7.

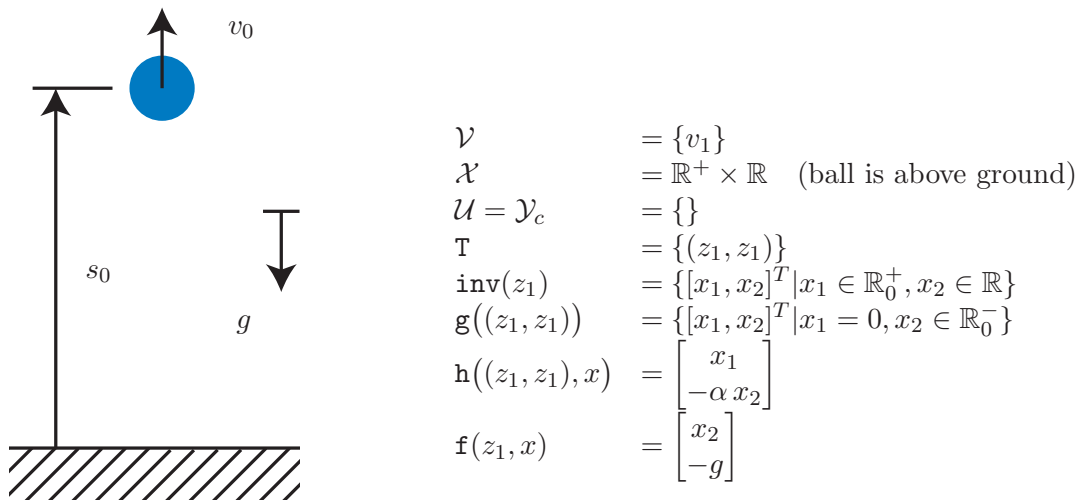


Figure 7: Bouncing ball.

The MATLAB code that implements the simulation and reachability analysis of the bouncing ball example is:

```
1 %%set reachability options
2 options.R0 = zonotope([[1; 0], diag([0.05, 0.05])]); %initial set
3 options.startLoc = 1; %initial location
4 options.finalLoc = 0; %0: no final location
5 options.tStart = 0; %start time
6 options.tFinal = 1.7; %final time
7 options.timeStepLoc{1} = 0.05; %time step size in location 1
8 options.taylorTerms = 10; %Taylor terms for exponential matrix
9 options.zonotopeOrder = 20; %maximum zonotope order
10 options.polytopeOrder = 10; %maximum order for conversion to polytopes
11 options.reductionTechnique = 'girard'; %reduction technique
12 options.enclosureEnables = 5; %choose enclosure method(s)
13 options.originContained = 0; %origin contained in input set?
14 options.isHyperplaneMap = 0; %hyperplane maps are not used
15 options.guardIntersect = 'polytope'; %convert guards to polytopes
16 options.uLoc{1} = 0; % no inputs
17 options.Uloc{1} = zonotope(0); % no inputs
18
19
20 %% set simulation options
21 simOptions.x0 = [1; 0]; %initial state for simulation
22 simOptions.startLoc = 1; %initial location
23 simOptions.finalLoc = 0; %0: no final location
24 simOptions.tStart = 0; %start time
25 simOptions.tFinal = 1.7; %final time
26 simOptions.uLoc{1} = 0; % no inputs
27 simOptions.Uloc{1} = zonotope(0); % no inputs
28
29
30 %% specify hybrid automaton
31 % convertd hybrid automaton model of the bouncing ball obtained from
32 % "spaceex2cora(bball.xml);"
33 HA = bball;
34
35 %simulate hybrid automaton
36 HA = simulate(HA,simOptions);
37
38 %compute reachable set
39 [HA] = reach(HA,options);
40
41 %% choose projection and plot
42 figure
43 hold on
44 options.projectedDimensions = [1 2];
45 options.plotType = 'b';
46 plot(HA,'reachableSet',options); %plot reachable set
47 plotFilled(options.R0,options.projectedDimensions,'w','EdgeColor','k'); %initial set
48 plot(HA,'simulation',options); %plot simulation
49 axis([0,1.2,-6,4]);
```

The reachable set and the simulation are plotted in Fig. 8 for a time horizon of $t_f = 1.7$.

8 Further Continuous Dynamics

These system classes are also implemented in CORA:

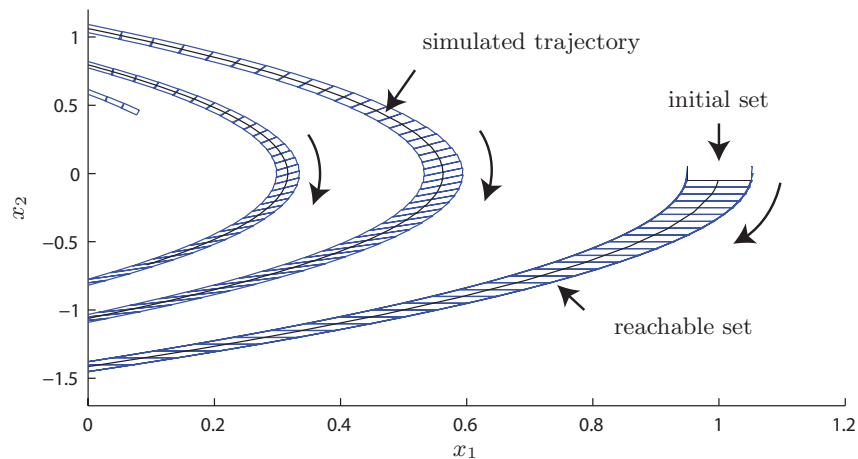


Figure 8: Illustration of the reachable set of the bouncing ball. The black box shows the initial set and the black line shows the simulated trajectory.

- Linear systems with uncertain parameters (constant parameters)
- Linear systems with uncertain parameters (time-varying parameters)
- Nonlinear systems with uncertain parameters (constant parameters)
- Nonlinear systems with uncertain parameters
- Nonlinear discrete time systems
- Differential-algebraic systems
- Parallel hybrid automata

A Troubleshooting

- Is the MATLAB path of CORA and the MPT toolbox set?
- Are all required MATLAB toolboxes installed?
- Are the unit tests all ok?
- Are variables reused? Use 'clear all'

References

- [1] M. Althoff. *Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars*. Dissertation, Technische Universität München, 2010. <http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20100715-963752-1-4>.
- [2] M. Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Hybrid Systems: Computation and Control*, pages 173–182, 2013.
- [3] M. Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, 2015.

- [4] M. Althoff and D. Grebenyuk. Implementation of interval arithmetic in CORA 2016. In *Proc. of the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 91–105, 2016.
- [5] M. Althoff, D. Grebenyuk, and N. Kochdumper. Implementation of Taylor models in CORA 2018. In *Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 145–173, 2018.
- [6] M. Althoff and B. H. Krogh. Zonotope bundles for the efficient computation of reachable sets. In *Proc. of the 50th IEEE Conference on Decision and Control*, pages 6814–6821, 2011.
- [7] Randal Beard. Quadrotor dynamics and control rev 0.1. Technical report, Brigham Young University, 2008.
- [8] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control*, LNCS 3414, pages 291–305. Springer, 2005.
- [9] J. Hoefkens, M. Berz, and K. Makino. *Scientific Computing, Validated Numerics, Interval Methods*, chapter Verified High-Order Integration of DAEs and Higher-Order ODEs, pages 281–292. Springer, 2001.
- [10] G. Lafferriere, G. J. Pappas, and S. Yovine. Symbolic reachability computation for families of linear vector fields. *Symbolic Computation*, 32:231–253, 2001.
- [11] J. K. Scott, D. M. Raimondo, G. R. Marseglia, and R. D. Braatz. Constrained zonotopes: A new tool for set-based estimation and fault detection. *Automatica*, 69:126–136, 2016.
- [12] O. Stursberg and B. H. Krogh. Efficient representation and computation of reachable sets for hybrid systems. In *Hybrid Systems: Computation and Control*, LNCS 2623, pages 482–497. Springer, 2003.
- [13] A. van der Schaft and H. Schumacher. *An Introduction to Hybrid Dynamical Systems*. Springer, 2000.