Verifying a PI Controller using SoapBox and Stabhyli[*]

Experiences on Establishing Properties for a Steering Controller

Willem Hagemann and Eike Möhlmann and Astrid Rakow
Carl von Ossietzky University of Oldenburg
Department of Computer Science
D-26111 Oldenburg, Germany
{willem.hagemann,eike.moehlmann,astrid.rakow}@informatik.uni-oldenburg.de

ABSTRACT

In the following we describe practical experiences on verifying a steering controller specification. The hybrid automaton implements a PI control rule and considers the vehicle's velocity as input from the environment. By combining the tools STABHYLI and SOAPBOX, we establish several safety and liveness properties for the steering controller, including convergence towards an equilibrium.

Keywords

Hybrid Systems; Automatic Verification; Stability; Safety

1. INTRODUCTION

In the following we describe practical experiences on establishing safety and liveness properties for a steering controller (SC). The steering controller has been specified as part of a case study [8, 4] on designing loosely coupled systems, where we examined an advanced driver assistance system (ADAS) controller. Our ADAS controller is made up of two loosely coupled subcomponents, a velocity controller and the SC. Only the latter will be considered in the sequel. Whereas in [8] the focus was on modeling the loosely coupled system within a design framework, we concentrate in this report on the verification steps and in particular on the techniques necessary to establish the properties with our tools STABHYLI and SOAPBOX [6]. Also we analyze only the SC here. Therefore we consider the SC as hybrid automaton with an input, that is the velocity of the vehicle. We give an overview on SC and the examined properties in Sect. 2. In Sect. 3 and Sect. 4 we then briefly introduce STABHYLI and SOAPBOX, the tools we used to establish the properties. Finally, in Sect. 5, we describe the actual process of establishing the properties, that is manual as well as (partially) automatized transformations of the model and proof steps of the verification process.

[*]This work has been partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

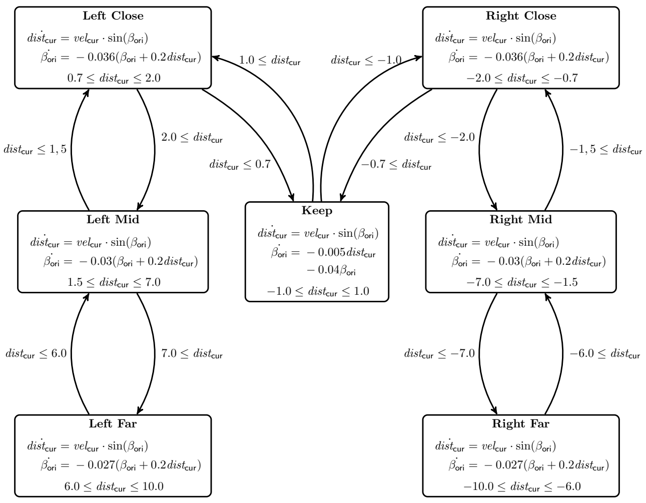

Left Close
$\dot{dist}_{cur} = vel_{cur} \cdot \sin(\beta_{ori})$
$\dot{\beta}_{ori} = -0.036(\beta_{ori} + 0.2 dist_{cur})$
$0.7 \le dist_{cur} \le 2.0$
Right Close
$\dot{dist}_{cur} = vel_{cur} \cdot \sin(\beta_{ori})$
$\dot{\beta}_{ori} = -0.036(\beta_{ori} + 0.2 dist_{cur})$
$-2.0 \le dist_{cur} \le -0.7$
$1.0 \le dist_{cur}$
$dist_{cur} \le -1.0$
$dist_{cur} \le 1,5$
$2.0 \le dist_{cur}$
$dist_{cur} \le 0.7$
$-0.7 \le dist_{cur}$
$dist_{cur} \le -2.0$
$-1,5 \le dist_{cur}$
Left Mid
$\dot{dist}_{cur} = vel_{cur} \cdot \sin(\beta_{ori})$
$\dot{\beta}_{ori} = -0.03(\beta_{ori} + 0.2 dist_{cur})$
$1.5 \le dist_{cur} \le 7.0$
Keep
$\dot{dist}_{cur} = vel_{cur} \cdot \sin(\beta_{ori})$
$\dot{\beta}_{ori} = -0.005 dist_{cur} - 0.04\beta_{ori}$
$-1.0 \le dist_{cur} \le 1.0$
Right Mid
$\dot{dist}_{cur} = vel_{cur} \cdot \sin(\beta_{ori})$
$\dot{\beta}_{ori} = -0.03(\beta_{ori} + 0.2 dist_{cur})$
$-7.0 \le dist_{cur} \le -1.5$
$dist_{cur} \le 6.0$
$7.0 \le dist_{cur}$
$dist_{cur} \le -7.0$
$-6.0 \le dist_{cur}$
Left Far
$\dot{dist}_{cur} = vel_{cur} \cdot \sin(\beta_{ori})$
$\dot{\beta}_{ori} = -0.027(\beta_{ori} + 0.2 dist_{cur})$
$6.0 \le dist_{cur} \le 10.0$
Right Far
$\dot{dist}_{cur} = vel_{cur} \cdot \sin(\beta_{ori})$
$\dot{\beta}_{ori} = -0.027(\beta_{ori} + 0.2 dist_{cur})$
$-10.0 \le dist_{cur} \le -6.0$

Figure 1: Steering Controller

2. STEERING CONTROLLER EXAMPLE

The steering controller can be thought of as one component within a composed more complex controller. SC is in charge of steering: it has the objective to bring the car to the center of its lane and then keep it there. The speed vel_{cur} of the car is an input or uncontrolled disturbance for steering. The current distance $dist_{cur}$ of the car with respect to the center of the lane evolves with $\dot{dist}_{cur} = vel_{cur} \cdot \sin(\beta_{ori})$. SC determines the change of the car's orientation $\dot{\beta}_{ori}$ as illustrated in Fig.1. Basically, SC performs steering control in a PI-like manner exploiting the distance as an integrator. It distinguishes three major operation modes Left, Right and Keep. In the latter the car is quite close to the mid of the lane, in modes Left and Right the car is further away from the mid. Both modes, Left and Right, are split up into three modes for distances classified as either close, mid or far, and parameters are tweaked for the respective distance category.

In [8, 4] we report on establishing that

(p1) SC converges to $(\beta_{ori}, dist_{cur}) = (0,0)$,
(p2) $\mathsf{G}\ (dist_{cur} \in [-10, 10] \wedge \beta_{ori} \in [-5^\circ, 5^\circ])$, and
(p3) $\mathsf{G}\ (\dot{\beta}_{ori} \in [-0.3, 0.3])$.

These properties are interface relevant properties of the SC component within our framework for library components. Before we describe how we established these properties, we briefly introduce the tools applied to establish them.

3. STABHYLI

In this section we briefly describe the tool Stabhyli [9]. Stabhyli can be used to obtain Lyapunov functions, that certify stability for hybrid systems. It handles non-linear systems whose behavior is expressible in forms of polynomials. Stabhyli can be used to obtain a single common Lyapunov function, a piecewise Lyapunov function as well as to perform the (de-)compositional proof schemes presented in [11, 3]. These features are fully automatized and combined with pre- and postprocessing steps that simplify the design and counteract numerical problems. Furthermore, in case stability cannot be proven, Stabhyli returns a hint to the user.

In the sequel we briefly sketch the theoretical basis of Stabhyli and introduce some fundamental notions. A hybrid automaton is defined as follows

Definition 1. A **Hybrid Automaton** is a quintuple

$$H = (\mathcal{V}, \mathbb{M}, R^{\mathsf{dscr}}, R^{\mathsf{cnt}}, \Theta^{\mathsf{inv}}, \varphi^{\mathsf{init}}) \text{ where}$$

- $\mathcal{V}$ is a finite set of *variables* and $\mathcal{S} = \mathbb{R}^{|\mathcal{V}|}$ is the corresponding *continuous state space*,
- $\mathbb{M}$ is a finite set of *modes*,
- R^{dscr} is a finite set of *transitions* (m_1, G, U, m_2) where
  - $m_1, m_2 \in \mathbb{M}$ are the *source and target mode* of the transition, respectively,
  - $G \subseteq \mathcal{S}$ is a *guard* which restricts the valuations of the variables for which this transition can be taken,
  - $U : \mathcal{S} \to \mathcal{S}$ is the *update function* which specifies discrete changes of variable valuations,
- $R^{\mathsf{cnt}} : \mathbb{M} \to [\mathcal{S} \to \mathcal{P}(\mathcal{S})]$ is the *flow function* which assigns a *flow* to every mode. A flow $f \subseteq \mathcal{S} \to \mathcal{P}(\mathcal{S})$ in turn assigns a closed subset of $\mathcal{S}$ to each $x \in \mathcal{S}$, which can be seen as the right hand side of a differential inclusion $\dot{x} \in f(x)$,
- $\Theta^{\mathsf{inv}} : \mathbb{M} \to \mathcal{S}$ is the *invariant function* which assigns a closed subset of the continuous state space to each mode $m \in \mathbb{M}$, and therefore restricts valuations of the variables for which this mode can be active, and
- $\varphi^{\mathsf{init}} \subseteq \mathbb{M} \times \mathcal{S}$ is a set of initial states.

A *trajectory* of H is an infinite solution in form of a function $(x(t), m(t))$ over time where $x(\cdot)$ describes the evolution of the continuous variables and $m(\cdot)$ the corresponding evolution of the modes.

Intuitively, stability is a property expressing that all trajectories of the system eventually reach an equilibrium point of the sub-state space and stay in that point forever given the absence of errors. For technical reasons the equilibrium point is usually assumed to be the origin of the continuous state space, i. e. $\mathbf{0}$. This is not a restriction, since a system can always be shifted such that the equilibrium is $\mathbf{0}$ via a coordinate transformation. In the sequel, we focus on *asymptotic stability* which does not require the equilibrium point to be reached in finite time but only requires every trajectory to "continuously approach" it (in contrast to *exponential stability* where additionally the existence of an exponential rate of convergence is required).

Definition 2. **Global Asymptotic Stability with Respect to a Subset of Variables [10].** Let $H = (\mathcal{V}, \mathbb{M}, R^{\mathsf{dscr}}, R^{\mathsf{cnt}}, \Theta^{\mathsf{inv}})$ be a hybrid automaton, and let $\mathcal{V}' \subseteq \mathcal{V}$ be the set of variables that are required to converge to the equilibrium point $\mathbf{0}$. A continuous-time dynamic system H is called *globally stable (GS) with respect to* $\mathcal{V}'$ if for all functions $x_{\downarrow\mathcal{V}'}(\cdot)$,

$$\forall \epsilon > 0 : \exists \delta > 0 : \forall t \geq 0 : ||x(0)|| < \delta \Rightarrow ||x_{\downarrow\mathcal{V}'}(t)|| < \epsilon.$$

H is called *globally attractive (GA) with respect to* $\mathcal{V}'$ if for all functions $x_{\downarrow\mathcal{V}'}(\cdot)$,

$$\lim_{t\to\infty} x_{\downarrow\mathcal{V}'}(t) = \mathbf{0}, \text{ i.e.,} \forall \epsilon > 0 : \exists t_0 \geq 0 : \forall t > t_0 : ||x_{\downarrow\mathcal{V}'}(t)|| < \epsilon,$$

where $\mathbf{0}$ is the origin of $\mathbb{R}^{|\mathcal{V}'|}$. If a system is both globally stable with respect to $\mathcal{V}'$ and globally attractive with respect to $\mathcal{V}'$, then it is called *globally asymptotically stable (GAS) with respect to* $\mathcal{V}'$.

In the following, we denote with $x_{\downarrow\mathcal{V}'} \in \mathbb{R}^{|\mathcal{V}'|}$ the sub-vector of vector $x \in \mathbb{R}^{|\mathcal{V}|}$ that represents the according valuation of $\mathcal{V}' \subseteq \mathcal{V}$.

Theorem 1. **Discontinuous Lyapunov Functions for a subset of variables [10].** Let $H = (\mathcal{V}, \mathbb{M}, R^{\mathsf{dscr}}, R^{\mathsf{cnt}}, \Theta^{\mathsf{inv}})$ be a hybrid automaton and let $\mathcal{V}' \subseteq \mathcal{V}$ be the set of variables that are required to converge. If for each $m \in \mathbb{M}$, there exists a set of variables $\mathcal{V}_m$ with $\mathcal{V}' \subseteq \mathcal{V}_m \subseteq \mathcal{V}$ and a continuously differentiable function $V_m : \mathcal{S} \to \mathbb{R}$ such that

1. for each $m \in \mathbb{M}$, there exist two class K^∞ functions α and β such that
$$\forall x \in \Theta^{\mathsf{inv}}(m) : \alpha(||x_{\downarrow\mathcal{V}_m}||) \preceq V_m(x) \preceq \beta(||x_{\downarrow\mathcal{V}_m}||),$$
2. for each $m \in \mathbb{M}$, there exists a class K^∞ function γ such that
$$\forall x \in \Theta^{\mathsf{inv}}(m) : \dot{V}_m(x) \preceq -\gamma(||x_{\downarrow\mathcal{V}_m}||)$$
for each $\dot{V}_m(x) \in \left\{ \left\langle \frac{dV_m(x)}{dx} \middle| f(x) \right\rangle \middle| f(x) \in R^{\mathsf{cnt}}(m) \right\}$,
3. for each $(m_1, G, U, m_2) \in R^{\mathsf{dscr}}$,
$$\forall x \in G : V_{m_2}(U(x)) \preceq V_{m_1}(x),$$

then H is globally asymptotically stable with respect to $\mathcal{V}'$ and V_m is called a *Local Lyapunov Function (LLF)* of m.

In Theorem 1, $\left\langle \frac{dV(x)}{dx} \middle| f(x) \right\rangle$ denotes the inner product between the gradient of a Lyapunov function V and a flow function $f(x)$.

Stabhyli now generates constraint systems using the so called *sums-of-squares* method [12] and the S-Procedure [2] to generate linear matrix inequalities which can then be solved by a *semi-definite program (SDP)*. To do this the numerical solver CSDP [1] is in charge. If such a constraint system is feasible then each solution represents a valid Lyapunov function.

4. SOAPBOX

This section briefly describes SoapBox, a tool for reachability analysis of hybrid systems, which is implemented in Matlab. SoapBox can handle hybrid systems with continuous dynamics described by linear differential inclusions and arbitrary linear maps for discrete updates. The invariants, guards, and sets of reachable states are given as convex polyhedra. The reachability algorithm of SoapBox is based on a novel representation class for convex polyhedra, the symbolic orthogonal projections (sops), on which various geometric operations, including convex hulls, Minkowski sums, linear maps, and intersections, can be performed efficiently and exactly. The capability to represent intersections of convex polyhedra exactly is superior to support function-based approaches like the LGG-algorithm (Le Guernic and Girard [7]), which is implemented in SpaceEx [5].

4.1 Symbolic Orthogonal Projections

A *symbolic orthogonal projection* (*sop*) $\mathbf{P} = \mathbf{P}(A, L, \mathbf{a})$ encodes the orthogonal projection of a higher dimensional $\mathcal{H}$-polyhedron $\mathbf{P}((A\ L), \mathbf{a}) = \left\{ \binom{\mathbf{x}}{\mathbf{z}} \middle| A\mathbf{x} + L\mathbf{z} \leq \mathbf{a} \right\}$ onto the vector space of dimension d:

$$\mathbf{P}(A, L, \mathbf{a}) = \left\{ \mathbf{x} \in \mathbb{K}^d \middle| \exists \mathbf{z} \in \mathbb{K}^k : A\mathbf{x} + L\mathbf{z} \leq \mathbf{a} \right\}.$$

Obviously, the following properties hold for sops: a sop $\mathbf{P}(A, L, \mathbf{a})$ is empty if and only if its preimage $\mathbf{P}((A\ L), \mathbf{a})$ is empty; any $\mathcal{H}$-polyhedron $\mathbf{P} = \mathbf{P}(A, \mathbf{a}) \in \mathbb{K}^d$ can be represented by the sop $\mathbf{P}(A, \varnothing, \mathbf{a})$, where $\varnothing$ denotes the empty matrix; and the support function $h_{\mathbf{P}}(\mathbf{n})$ is given by the optimal value of the linear program

$$\text{maximize } \mathbf{n}^T\mathbf{x} \text{ subject to } A\mathbf{x} + L\mathbf{z} \leq \mathbf{a}.$$

The representation of polyhedra as orthogonal projections of higher dimensional polyhedra provides the freedom to introduce existentially quantified variables. For example, the Minkowski sum of two polyhedra $\mathbf{P}_1 = \mathbf{P}(A_1, \mathbf{a}_1)$ and $\mathbf{P}_2 = \mathbf{P}(A_2, \mathbf{a}_2)$ is defined as the set

$$\{\mathbf{z} | \exists \mathbf{x}, \mathbf{y} : A_1\mathbf{x} \leq \mathbf{a}_1, A_2\mathbf{y} \leq \mathbf{a}_2, \mathbf{z} = \mathbf{x} + \mathbf{y}\}.$$

Now, the Minkowski sum can be rewritten to

$$\begin{aligned} &\left\{\mathbf{z} \middle| \exists \mathbf{y}' : A_1(\mathbf{z} + \mathbf{y}') \leq \mathbf{a}_1, -A_2\mathbf{y}' \leq \mathbf{a}_2\right\} \\ &= \mathbf{P}\left(\begin{pmatrix} A_1 \\ \mathrm{O} \end{pmatrix}, \begin{pmatrix} A_1 \\ -A_2 \end{pmatrix}, \begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{pmatrix}\right). \end{aligned}$$

By introduction of certain existentially quantified variables we obtain exact and efficient sop-based representations of the closed convex hulls, Minkowski sums, intersections and arbitrary linear transformations of polyhedra. In fact, most of these operations are realized as block matrices over the original representation matrices.

Overapproximation of Sops

Due to the introduction of additional variables the size of the representation matrices of the assembled sops grows monotonically. Hence, the evaluation of assembled sops by means of linear programming gets increasingly harder. The overapproximation of a sop by an $\mathcal{H}$-polyhedron with a fixed representation matrix (*template polyhedron*) helps to shrink the size but might induce a substantial loss of the facial structure of the sop. To recover at least some of the facial structure we use a post-processing technique called ray shooting: Let $\mathbf{P} = \mathbf{P}(A, L, \mathbf{a})$ be a non-empty sop in $\mathbb{K}^d$ which contains the origin $\mathbf{0}$ as a relative interior point and $\mathbf{r}$ be the direction of some ray $\lambda\mathbf{r}$ in $\mathbb{K}^d$, $\lambda \geq 0$. If $\mathbf{P}$ is neither unbounded nor flat in direction $\mathbf{r}$, i. e. a maximal $\lambda_0 > 0$ with $\lambda_0\mathbf{r} \in \mathbf{P}$ exists, then there is a linear program which provides a normal vector $\mathbf{n}$ of a supporting half-space $\mathbf{H} = \mathbf{H}(\mathbf{n}, 1)$ of $\mathbf{P}$ containing the boundary point $\lambda_0\mathbf{r}$.

4.2 Reachability Analysis Using Sops

Internally, SoapBox uses *symbolic states* to represent the reachable sets of the hybrid system. A *symbolic state* is a pair $(\mathbf{P}, m)$ of a polyhedron $\mathbf{P} \subseteq \mathcal{S}$ and a mode $m \in \mathbb{M}$. The reachable states from a symbolic state $(\mathbf{P}, m)$ are exactly those states which are reached by a trajectory emanating from $(\mathbf{P}, m)$.

Given a hybrid automaton H and the sets *Init*, *Safe*, and *Unsafe* as disjunction of finitely many symbolic states, the goal of SoapBox is to compute all reachable states from the initial states in *Init* until the trajectory enters *Safe*, touches *Unsafe*, or leaves the mode invariant $\Theta^{\mathsf{inv}}(m)$.

Let $(\mathbf{P}, m)$ be a symbolic state. Then the reachable states of $(\mathbf{P}, m)$ within the mode m are described by the linear system

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \mathbf{E} \in R^{\mathsf{cnt}},\ \mathbf{x}(0) \in \mathbf{P},\ \mathbf{x} \in \Theta^{\mathsf{inv}}(m),$$

where $\mathbf{E}$ is a convex polytope modelling the bounded input. A discrete transition $(m, \mathbf{G}, \mathbf{U}, m') \in R^{\mathsf{dscr}}$ can be taken for all solutions $\mathbf{x}(t)$ which satisfy the guard $\mathbf{G}$, i. e. $\mathbf{x}(t) \in \mathbf{G}$.

SoapBox computes an overapproximation of the reachable states in a step-wise manner. Let δ be a time-step parameter. First, a polyhedral overapproximation of the reachable states within the time-interval $[0, \delta]$ is computed using a safe first-order approximation of the continuous dynamics (initial bloating). The initial bloating procedure yields two sops, a bloated set $\mathbf{R}_0$ which contains all reachable states within the time-interval $[0, \delta]$ and a sop $\mathbf{V}$ which accounts for the influences of the bounded input $\mathbf{E}$ within the time-interval $[0, \delta]$. Now, the reachable states within the time-interval $[k\delta, (k+1)\delta]$ are computed according to the recursive formula

$$\mathbf{R}_{k+1} = (\mathrm{e}^{\delta A}(\mathbf{R}_k) + \mathbf{V}) \cap \Theta^{\mathsf{inv}}(m).$$

Then all intersections with possible guards are computed until the current flow segment leaves the invariant or completely lies in a symbolic *Safe* state. In the next step, SoapBox checks whether one of the resulting intersections is an intersection with one of the symbolic *Unsafe* states. In this case SoapBox stops the reachability analysis with an appropriate output. Otherwise, the discrete post-images of the guard intersections are computed and added to the initial states.

In fact, SoapBox combines the sop-based reachability computation with the reachability algorithm of Le Guernic and Girard (LGG-algorithm) [7], which computes a coarser overapproximation of the reachable states. While the sop-based part of the algorithm has the capability to provide an exact representation, the LGG-algorithm yields fast overapproximations in terms of template polyhedra, e. g. it almost completely ignores the influence of the invariant. Those operations of the reachability analysis which involve linear programming, e. g. the subset check for safe sets or the intersection tests for guards, are performed in the LGG-part. After a given number of computation steps, SoapBox uses ray shooting to improve the overapproximation computed in the LGG-part by adding some additional information of the facial structure gained by the sop part. Then the current sop is replaced by this overapproximation. This procedure efficiently shrinks the representations size of the sops, but leads to a loss of accuracy. Anyhow, since this method may only improve the state sets generated in LGG-part, Soap-

Box still archives tighter overapproximations than the pure LGG-algorithm.

We should note that the current implementation of SoapBox internally uses floating point numbers. While most sop-based operations consist of block matrix operations and hence do not introduce additional numerical issues, there are two main sources of numerical issues regarding the use of floating point numbers: the usage of the matrix exponential $e^{\delta A}$ and the usage of the inexact linear solver Gurobi. Anyhow, SoapBox has only a few built-in functions, like scaling, to deal with potential numerical issues. Instead, the numerical values are passed over to the linear solver. Experiments with GLPK and Gurobi showed that Gurobi copes quite well with potential numerical issues. The numerical issues stemming from the usage of an inexact solver can be fixed by using exact arithmetic. Theoretically more challenging, is a numerical safe evaluation of the matrix exponential, which also SpaceEx has to face. This issue is regarded as a promising future research direction.

Handling Strict Inequalities

In order to handle linearizations of dynamics with non-linear differential inclusions properly, we also allow guards with strict inequalities. SoapBox handles strict inequalities in the following way: A transition involving a strict guard $\mathbf{n}^T\mathbf{x} < c$ is disabled as long as the current flow segment $\mathbf{P}$ does not contain a witness point $\mathbf{x}$ which satisfies the strict guard. Otherwise, the transition is enabled and we treat the strict inequality as a non-strict inequality. Hence, in this case, we compute a closed overapproximation of the actual intersection. The special treatment of strict inequalities avoids zeno-behavior which would occur if we handled strict inequalities as non-strict inequalities: Let x be a variable of a hybrid automaton with the two modes m_1 and m_2. Further, let $(m_1, x > 5, id, m_2)$ and $(m_2, x < 5, id, m_1)$ be two discrete transitions. Treating the strict inequalities as non-strict inequalities would allow zeno-behavior for $x = 5$. With our treatment of the strict inequalities, the transition from m_1 to m_2 is enabled for any set (m_1, X) containing at least one element $\mathbf{x} > 5$. The post-image of (m_1, X) is $(m_2, X \cap \{x | x \geq 5\})$ and does not contain a witness for the guard $x < 5$. Hence, an instantaneous transition back from m_2 to m_1 is disabled and zeno-behavior is avoided.

For such cases this guarantees progress in the reach set computation.

5. PROOF STEPS

Up to now, we introduced the SC model, the properties we are interested in (cf. Sect. 2) and the tools we are going to apply (cf. Sect. 3, 4). In the following we describe the steps of establishing properties (p1) to (p3). Basically, for successful verification we had to (i) bridge the model-tool gap, (ii) simplify the model, (iii) simplify the proof goals and (iv) apply the tools.

5.1 Proving (p1)

First we showed that SC converges to $(\beta_{\mathsf{ori}}, dist) = (0, 0)$ using Stabhyli. In order to apply Stabhyli we used a simple overapproximation of the sine and Stabhyli was able to certify convergence for the overapproximated system.

In the following proof steps we could greatly benefit from having established convergence first. In particular we used the Lyapunov functions (LFs) obtained from Stabhyli to alleviate the proof obligations for establishing (p2).

5.2 Proving (p2)

To reduce the automaton's complexity we used the observation that by design SC is symmetric. This yields a steering controller where either the left or right modes are omitted (cf. Fig. 2).

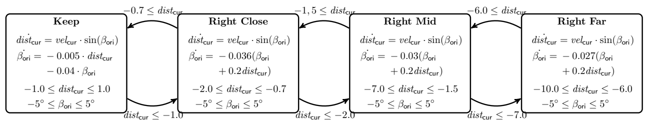

Keep
$\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.005 \cdot dist_{\mathsf{cur}} - 0.04 \cdot \beta_{\mathsf{ori}}$
$-1.0 \leq dist_{\mathsf{cur}} \leq 1.0$
$-5° \leq \beta_{\mathsf{ori}} \leq 5°$
Right Close
$\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.036(\beta_{\mathsf{ori}} + 0.2 dist_{\mathsf{cur}})$
$-2.0 \leq dist_{\mathsf{cur}} \leq -0.7$
$-5° \leq \beta_{\mathsf{ori}} \leq 5°$
Right Mid
$\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.03(\beta_{\mathsf{ori}} + 0.2 dist_{\mathsf{cur}})$
$-7.0 \leq dist_{\mathsf{cur}} \leq -1.5$
$-5° \leq \beta_{\mathsf{ori}} \leq 5°$
Right Far
$\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.027(\beta_{\mathsf{ori}} + 0.2 dist_{\mathsf{cur}})$
$-10.0 \leq dist_{\mathsf{cur}} \leq -6.0$
$-5° \leq \beta_{\mathsf{ori}} \leq 5°$
$-0.7 \leq dist_{\mathsf{cur}}$
$-1,5 \leq dist_{\mathsf{cur}}$
$-6.0 \leq dist_{\mathsf{cur}}$
$dist_{\mathsf{cur}} \leq -1.0$
$dist_{\mathsf{cur}} \leq -2.0$
$dist_{\mathsf{cur}} \leq -7.0$

Figure 2: SC after Exploiting Symmetry

We also transformed the proof goal. If any trajectory of the reduced model emanating from the initial set finally re-enters the initial set in mode **Keep**, we can verify (p2) (and also (p3)) on the reduced model, as the omitted part of the automaton does not inject additional behavior in the retained part.

In general, we can stop computation of a trajectory, when it re-enters the initial set. Any possible suffix of it is already covered as a trajectory directly emanating from the initial set.

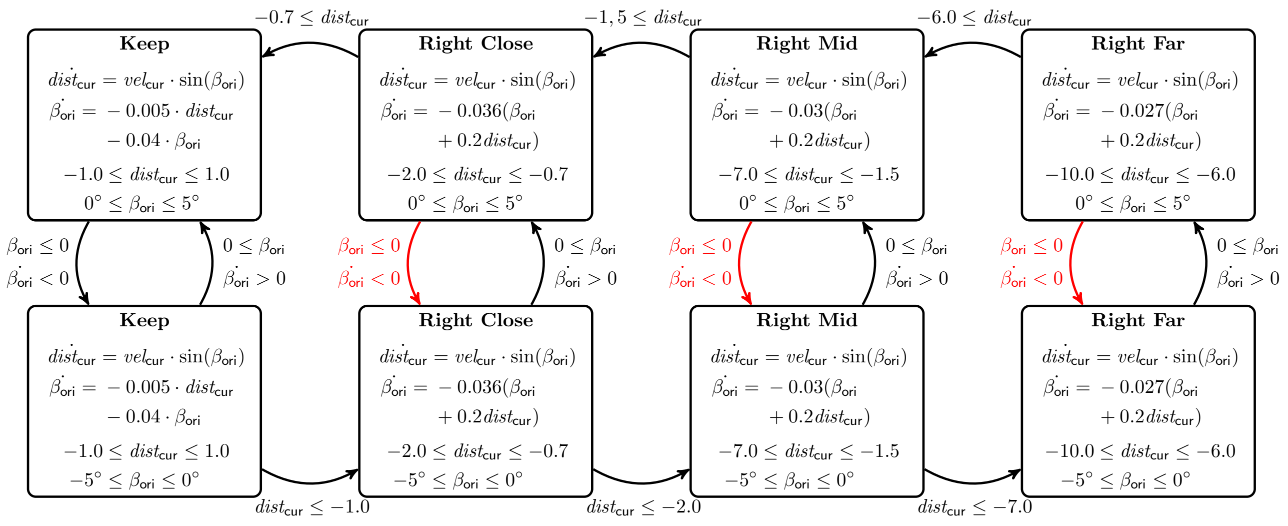
In essence, we show on the reduced model that 1. any trajectory emanating from the initial set re-enters the initial set and 2. any reachable state along a trajectory from initial set up to re-entering the initial set satisfies (p2).

To use SoapBox for verification of SC we have to transform the model, so that 1. the bounded input vel_{cur} is additive instead of multiplicative, 2. the sinus function is overapproximated. To get rid of the sine, we overapproximated the distance evolution $\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$ for $vel_{\mathsf{cur}} \in [5, 10]$. More precisely, we overapproximated $\dot{dist}_{\mathsf{cur}} \in \text{convex}(5 \cdot \sin(\beta_{\mathsf{ori}}), 10 \cdot \sin(\beta_{\mathsf{ori}}))$ by $\dot{dist}_{\mathsf{cur}} = a \cdot \beta_{\mathsf{ori}} + E$ where E is a convex polytope denoting a bounded error. Since a single overapproximation over all possible values, i.e., $\beta_{\mathsf{ori}} \in [-5°, 5°]$, turned out to be too coarse, we partitioned the state space along β_{ori}[1]. This yields a better fitting overapproximation, but as a side effect the SC's modes are split up, as illustrated in Figure 3. Drawbacks of such partitioning hence are, that a lot of new transitions are introduced and some of them can even lead to zeno-behavior. To prove (p2) we used a finer partition which results in a large automaton. It is easy to see that some of its transitions do not correspond to the original flow of the system and can hence be removed (marked in red in Figure 3).

To apply a partition, SoapBox has to deal with strict inequations. Strict inequations allow us to express that subsets are disjoint, while the complete state space is covered. A division into disjoint subset allows finer overapproximation. As discussed in Sect. 4.2 SoapBox supports strict inequations and thereby rules out some forms of zeno-behavior. Note that transition removal described above was not applicable to the **Keep** mode.

We further simplified the proof goal. To show the safety property (p2), we have to show that all reachable states

[1]Actually the division is overlapping for the boundary value of β_{ori} and transitions refer to $\dot{\beta}_{\mathsf{ori}} < 0$ or $\dot{\beta}_{\mathsf{ori}} > 0$, respectively.



$-0.7 \leq dist_{\text{cur}}$
$-1,5 \leq dist_{\text{cur}}$
$-6.0 \leq dist_{\text{cur}}$
Keep
$\dot{dist}_{\text{cur}} = vel_{\text{cur}} \cdot \sin(\beta_{\text{ori}})$
$\dot{\beta}_{\text{ori}} = -0.005 \cdot dist_{\text{cur}} - 0.04 \cdot \beta_{\text{ori}}$
$-1.0 \leq dist_{\text{cur}} \leq 1.0$
$0° \leq \beta_{\text{ori}} \leq 5°$
Right Close
$\dot{dist}_{\text{cur}} = vel_{\text{cur}} \cdot \sin(\beta_{\text{ori}})$
$\dot{\beta}_{\text{ori}} = -0.036(\beta_{\text{ori}} + 0.2 dist_{\text{cur}})$
$-2.0 \leq dist_{\text{cur}} \leq -0.7$
$0° \leq \beta_{\text{ori}} \leq 5°$
Right Mid
$\dot{dist}_{\text{cur}} = vel_{\text{cur}} \cdot \sin(\beta_{\text{ori}})$
$\dot{\beta}_{\text{ori}} = -0.03(\beta_{\text{ori}} + 0.2 dist_{\text{cur}})$
$-7.0 \leq dist_{\text{cur}} \leq -1.5$
$0° \leq \beta_{\text{ori}} \leq 5°$
Right Far
$\dot{dist}_{\text{cur}} = vel_{\text{cur}} \cdot \sin(\beta_{\text{ori}})$
$\dot{\beta}_{\text{ori}} = -0.027(\beta_{\text{ori}} + 0.2 dist_{\text{cur}})$
$-10.0 \leq dist_{\text{cur}} \leq -6.0$
$0° \leq \beta_{\text{ori}} \leq 5°$
$\beta_{\text{ori}} \leq 0$ $\dot{\beta}_{\text{ori}} < 0$
$0 \leq \beta_{\text{ori}}$ $\dot{\beta}_{\text{ori}} > 0$
Keep
$-5° \leq \beta_{\text{ori}} \leq 0°$
Right Close
$-5° \leq \beta_{\text{ori}} \leq 0°$
Right Mid
$-5° \leq \beta_{\text{ori}} \leq 0°$
Right Far
$-5° \leq \beta_{\text{ori}} \leq 0°$
$dist_{\text{cur}} \leq -1.0$
$dist_{\text{cur}} \leq -2.0$
$dist_{\text{cur}} \leq -7.0$

Figure 3: SC after partitioning β_{ori} at 0

satisfy $dist_{\text{cur}} \in [-10, 10]$ and $\beta_{\text{ori}} \in [-5°, 5°]$. We used two different arguments that allowed us to derive that all future states of a trajectory satisfy (p2).

The first argument refers to the automaton. Analysis on the vector field reveal that (a) $dist_{\text{cur}} \in [-10, 10] \wedge \beta_{\text{ori}} \geq 2$ implies $\dot{\beta}_{\text{ori}} \leq 0$ and (b) $\beta_{\text{ori}} \geq 0$ implies $\dot{dist}_{\text{cur}} \geq 0$. Thus we can conclude that the upper copies of **RightClose**, **RightMid**, and **RightFar** in Figure 3 are safe wrt (p2) if (p2) holds on entering these modes. We instead established only that any trajectory that enters these modes already satisfies (p2).

The second argument introduces so-called *safe sets* that are subsets of the state space and guarantee that any trajectory entering such a set will not reach an *unsafe set* anymore.

Level Sets and Safe Sets

Systems of damped oscillating behavior like the SC are often difficult to deal with when it comes to determining their trajectories near the equilibrium due to numerical issues. In the remainder of the section we sketch an argument that alleviates the burden of examining every reachable state wrt a safety property by a sound argument based on the system's Lyapunov function. Basically, the argument allows us to neglect the future of a trajectory as soon as the trajectory is sufficiently close to the equilibrium. We further describe its concrete application on the case study to establish (p2).

A Lyapunov function assigns a value to any state of the state space and this LF's value along any trajectory decreases. Each *level set* $\mathcal{L}_{V,s} = \{x | V(x) \leq s\}$ is hence an invariant set as defined in Def. 3.

Definition 3. **Invariant Set.** A set of states $\mathcal{S}' \subseteq \mathcal{S}$ is called an *invariant set* of a hybrid automaton H if for all trajectories $\mathbf{x}(t)$ holds $\mathbf{x}(0) \in \mathcal{S}' \Rightarrow \forall t \geq 0 : \mathbf{x}(t) \in \mathcal{S}'$.

Now, the alleviating argument for establishing (p2) is, as soon as a level set has been entered that satisfies (p2), we can stop further computing the trajectory. As we want to stop trajectory computation as early as possible, we try to find a level set as big as possible. Therefore we determine the minimal LF's value over all states violating (p2). A state with a lesser LF's value is hence guaranteed not to be in the *unsafe set*. Such a state is element of the *safe set* (cf. Def. 4).

Definition 4. **Safe Set.** Given a set of states $Unsafe \subseteq \mathcal{S}$, a set $Safe_{Unsafe} \subseteq \mathcal{S}$ is called a *safe set* wrt $Unsafe$ for a hybrid automaton H, if for all trajectories $\mathbf{x}(t)$ holds $\mathbf{x}(0) \in Safe_{Unsafe} \Rightarrow \forall t \geq 0 : \mathbf{x}(t) \notin Unsafe$.

Note, that any level set $\mathcal{L}_{V,s}$, which has no intersection with the set $Unsafe$, serves as a safe set $Safe_{Unsafe}$.

To use safe sets for trajectory truncation in a tool, we have to be able to express when a state is element of the safe set, i. e. when its LF's value is sufficiently low. We obtained quadratic Lyapunov functions for SC from STABHYLI. For tools supporting linear constraints only, we underapproximate safe sets via a polyhedron.

To summarize, we used safe sets to truncate trajectory computation. As SOAPBOX supports linear constraints only, we underapproximated the safe set via safe boxes. This yields the following basic algorithm:

1. Determine the lowest LF's value $b := \min\{V(x) \,|\, x \in Unsafe\}$ of all states in the unsafe set.
2. Subtract a safety margin ϵ on the LF's value, $g := b - \epsilon$, to build the safe set.
3. Guess a box within the level set $\mathcal{L}_{V,g}$.

Such a box lies within the safe set, if the LF's value for any of its corner points is less then or equal to g. Multiple safe boxes can be found by applying multiple optimization functions such as maximizing the length of an edge or the length of the diagonal. This procedure can be automatized and multiple optimization functions can be combined. We achieved the best results on the SC model via manual optimization, though. This was easily feasible, as we had to deal with two dimensions only. The result is shown in Figure 4.

In principle, safe boxes can be encoded into the model. As SOAPBOX and many other tools require convex modes, a safe box within a mode hence implies that the surrounding of the safe box has to be split into several convex submodes. SOAPBOX supports convex safe sets directly, though, as outlined in Sect. 4.2.

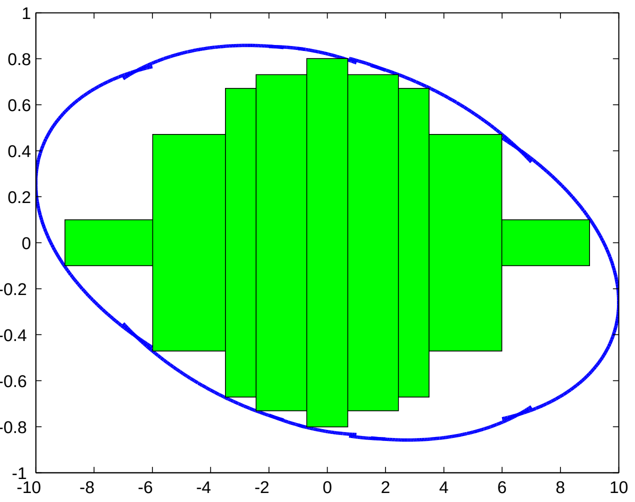


Figure 4: Piecewise Level Sets (blue) and Safe Sets (green) of the Steering Controller

Proving (p3)

That (p3) holds can be seen by examining the differential equation defining the evolution of β_{ori} as shown in Figure 5. The figure shows a heat map for all combinations of ($dist_{\text{cur}}$,β_{ori}) within the bounds as established by (p2). The color encodes the value of the differential equation for steering. It hence follows that the value of the differential equation ranges over $[-0.2, 0.2]$ and thus implies (p3) as long as (p2) holds.

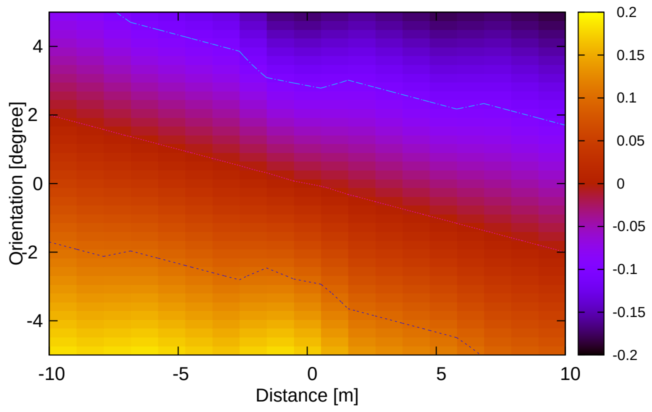


Figure 5: Examination of $\dot{\beta}_{\text{ori}}$

6. CONCLUSION

We have shown how to combine overapproximation, state space partitioning, Lyapunov functions, bounded reachability, and safe sets to prove non-trivial properties. Showing that a controller stabilizes without exceeding certain operation ranges seems to be an important verification task. While we believe that our steering controller is far from being an industrial-sized benchmark, it already shows that different techniques have to be combined to prove interesting properties.

7. REFERENCES

[1] B. Borchers. Csdp, a C library for semidefinite programming. 10:613–623, 1999.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Mar. 2004.

[3] W. Damm, H. Dierks, J. Oehlerking, and A. Pnueli. Towards Component Based Design of Hybrid Systems: Safety and Stability. In *Essays in Memory of Amir Pnueli*, volume 6200 of *LNCS*, pages 96–143. Springer, 2010.

[4] W. Damm, W. Hagemann, E. Möhlmann, and A. Rakow. Component based design of hybrid systems: A case study on concurrency and coupling. Reports of SFB/TR 14 AVACS 95, 2014.

[5] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395, 2011.

[6] W. Hagemann. Reachability analysis of hybrid systems using symbolic orthogonal projections. *CAV*, 2014. submitted.

[7] C. Le Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *Computer Aided Verification*, pages 540–554, 2009.

[8] E. Möhlmann, A. Rakow, and W. Damm. Component based design of hybrid systems: A case study on concurrency and coupling. In *HSCC*, 2014.

[9] E. Möhlmann and O. E. Theel. Stabhyli: a Tool for Automatic Stability Verification of Non-Linear Hybrid Systems. In C. Belta and F. Ivancic, editors, *HSCC*. ACM, 2013.

[10] J. Oehlerking. *Decomposition of Stability Proofs for Hybrid Systems*. PhD thesis, Carl von Ossietzky University of Oldenburg, Department of Computer Science, Oldenburg, Germany, 2011.

[11] J. Oehlerking and O. E. Theel. Decompositional Construction of Lyapunov Functions for Hybrid Systems. In *HSCC*, volume 5469 of *LNCS*, pages 276–290. Springer, 2009.

[12] S. Prajna and A. Papachristodoulou. Analysis of Switched and Hybrid Systems - Beyond Piecewise Quadratic Methods, 2003.