A Framework for Identification and Validation of Affine Hybrid Automata from Input-Output Traces

XIAODONG YANG, Vanderbilt University, USA
OMAR ALI BEG, The University of Texas Permian Basin, USA
MATTHEW KENIGSBERG, Vanderbilt University, USA
TAYLOR T JOHNSON, Vanderbilt University, USA

Automata-based modeling of hybrid and cyber-physical systems (CPS) is an important formal abstraction amenable to algorithmic analysis of its dynamic behaviors, such as in verification, fault identification, and anomaly detection. However, for realistic systems, especially industrial ones, identifying hybrid automata is challenging, due in part to inferring hybrid interactions, which involves inference of both continuous behaviors, such as through classical system identification, as well as discrete behaviors, such as through automata (e.g., L*) learning. In this paper, we propose and evaluate a framework for inferring and validating models of deterministic hybrid systems with linear ordinary differential equations (ODEs) from input/output execution traces. The framework contains algorithms for the approximation of continuous dynamics in discrete modes, estimation of transition conditions, and the inference of automata mode merging. The algorithms are capable of clustering trace segments and estimating their dynamic parameters, and meanwhile, deriving guard conditions that are represented by multiple linear inequalities. Finally, the inferred model is automatically converted to the format of the original system for the validation. We demonstrate the utility of this framework by evaluating its performance in several case studies as implemented through a publicly available prototype software framework called HAutLearn and compare it with a membership-based algorithm.

CCS Concepts: • **Computing methodologies** → **Modeling methodologies**; *Model verification and validation*; • **Computer systems organization** → Embedded and cyber-physical systems.

Additional Key Words and Phrases: Hybrid systems, System identification, Automata learning.

ACM Reference Format:
Xiaodong Yang, Omar Ali Beg, Matthew Kenigsberg, and Taylor T Johnson. 2020. A Framework for Identification and Validation of Affine Hybrid Automata from Input-Output Traces. In *Proceedings of* . ACM, New York, NY, USA, 25 pages.

1 INTRODUCTION

Modeling of systems has a long and storied history, with some of the original effective algorithms for finding automata from their traces described by Angluin's L* algorithm [4] now instantiated in software packages such as LearnLib [43]. From a software engineering vantage point, one can view finding automata from traces as a form of *specification inference* for an implementation of a system. Specification inference is an effective technique for automated documentation, model validation, model repair, and many other tasks, often restricted to subclasses of possible specifications that may be inferred, such as invariants [15–17]. From the automata theoretic perspective, one can thus view learning automata from traces as a method to infer classes of specifications beyond safety and into temporal behaviors, such as liveness.

Manuscript submitted to ACM

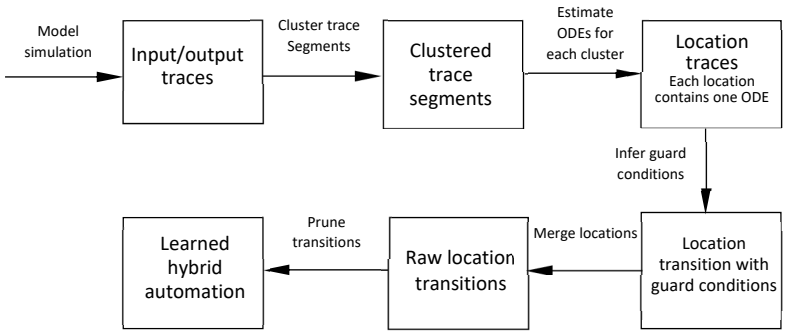

Fig. 1. Overview of the framework proposed in this paper to learn hybrid automata with linear (affine) ODEs, guards, invariants, and resets from time-series data (traces).

A hybrid automaton is a formal model that models both continuous and discrete behaviors through a combination of continuously-evolving real-valued variables and discrete components, which together exhibit mixed dynamical behaviors [25]. Continuous variables evolve over intervals of real time with respect to some specified ordinary differential equations (ODEs) or inclusions. Discrete behaviors are modeled in an automata-theoretic manner, typically defined by some forms of graphs or state machines. The discrete modes may contain invariant conditions which, once violated, will induce transitions to different modes. The transitions between discrete modes may also have guards with conditions including exogenous events and predicates over continuous variables. One of the canonical examples is the model of a bouncing ball. Released from a specified height, the ball exhibits different continuous dynamics after impacting the ground. Hybrid automata provides an expressive and useful abstraction to model different dynamical systems, and have proven valuable in various areas, such as system simulation, anomaly detection, reachability analysis, verification, and identification of optimal policies [3, 26]. Realistic systems are often too complex to be designed purely in a formalism such as hybrid automata, and often rely on complex software toolchains such as the MathWorks' Simulink/Stateflow, with a significantly more expressive modeling framework, but with unclear semantics. As hybrid automata models often are not the design engineer's modeling tool of choice, inferring hybrid automata from traces of complex and black-box systems can provide insight into the behaviors of those systems. Actual physical environments are usually too complicated to be analyzed using available technologies. Learning hybrid automata from system behaviors can provide a convenient way for system analysis in the abstraction layer so that the complexity of hybrid systems can be reduced while safety properties are still kept in its relevant behavior and the system itself can become accessible to existing analysis tools. Therefore, it helps engineers develop high-level automata strategies.

The contributions of this paper are that an automata-based framework for the inference and validation of hybrid systems from execution traces is developed, with restrictions on the continuous behaviors, guards, invariants, and resets to be described by *linear* (affine) equations. The inference framework includes five steps as shown in Figure 1: (1) cluster execution trace segments according to their dynamics, (2) fit an ordinary differential equation (ODE) to each cluster, (3) estimate guard conditions for the discontinuities, or *changepoints* in traces, (4) merge modes and transitions in terms of a defined compatibility criterion, and (5) prune duplicate and other erroneous transitions that arise from steps 1 and 3. We approximate the continuous dynamics by fitting linear (affine) ODEs to segmented traces, which is a classical problem in *system identification* [32]. To distinguish different dynamics from traces, we develop a method to calculate

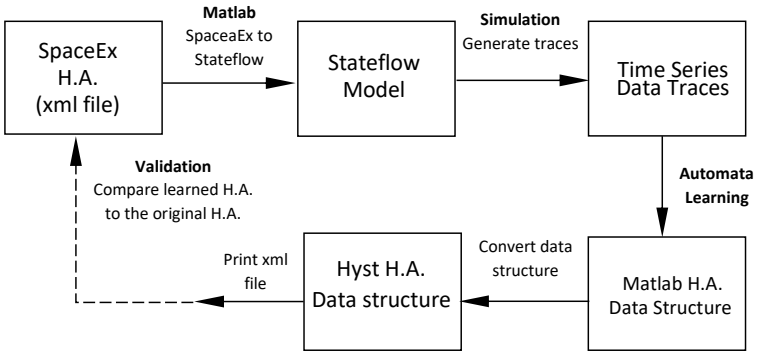

Fig. 2. Overview of the validation of the proposed framework for the method described in this paper. A given hybrid automaton is specified in the SpaceEx/HyST [7, 21] format, automatically translated to a Simulink/Stateflow model using HyST [6, 7], simulated to generate time series data (traces). The hybrid automata learning framework is then applied to these traces, and a learned hybrid automaton is generated in the SpaceEx/HyST format, and simulated again to generate traces to validate the learned model's behaviors against the original model's behaviors.

their solution spaces within a pre-specified error bound and cluster them accordingly. For guard conditions, under the assumption that they are described by linear inequalities, a subspace clustering algorithm is applied to estimate their parameters by clustering changepoints into a low-dimensional line or plane. For the mode merging, a method based on the prefix tree acceptor (PTA) is applied to merge similar modes without introducing non-determinism into the model, and then erroneous transitions are pruned before generating the final hybrid automaton that can recreate the source trace data. The framework is implemented in a prototype software tool within Matlab relying in part on the HyST source transformation and translation tool [7] and its integration with Simulink/Stateflow [6]. The framework is evaluated and validated against several standard hybrid systems benchmarks. These examples were chosen in part so that the validation approach illustrated in Figure 2 could be illustrated, where both syntactic and semantic similarities could be compared to the learned automata.

2 RELATED WORK

Significant related works have been developed in the context of automata learning for *purely discrete* systems, such as finite state automata [4]. For timed, switched, and hybrid systems, there has been less investigation, although there are several recently proposed methods [35, 42, 47, 49]. From the control theory, there are more related works for the system identification, including the identification of piecewise models such as the Switched affine AutoRegressive eXogenous (SARX) model and the PieceWise affine ARX (PWARX) model. A primary challenge of such identification includes the inference of the parameters of all potential models as well as the coefficients of the hyperplanes that partition the state-input domain. Such approaches have been well studied [22, 31, 41]. The majority of works can be divided into three categories: algebraic based [9, 33, 50–52], clustering-based [10, 13, 19, 23, 27], and optimization based [8, 28, 29, 36, 40]. The algebraic methods regard the identification of multi-models as one single model. The parameters are estimated with a polynomial embedding from whose derivatives the original model can be estimated. The clustering-based methods utilize feature vectors computed from local data sets to cluster the models and then estimate their parameters. The optimization-based methods convert the estimation of models to an optimization problem such as minimization of a

predefined loss function. Most of these works mainly focus on the identification of models, but few of them consider the inference of the overall automaton and switching policies.

Recent works on identification of hybrid automata are as follows. Summerville et al. propose a framework for hybrid automaton inference [46], where a cost function with a penalty criterion based on one set of potential linear model templates is applied to segment the traces and then select an optimal model with the minimum trace error. Their guard conditions are selected from predefined predicates using Normalized Pointwise Mutual Information (NPMI). Niggemann et al. model each segment using linear regression or neural networks [39]. The similarity of states is tested by checking the probability of staying in the state. States are merged in a bottom-up fashion. Guard conditions are estimated by a combination of exogenous events, timing constraints, and transition probabilities. Medhat et al. cluster the observed traces into input/output events according to predefined features [35], and then apply the linear regression to estimate the clustered dynamics. They derive an automaton based on a Mealy inference algorithm within LearnLib. Grosu et al. propose a methodology to estimate cycle-linear hybrid automata for excitable cells from virtual measurements [24]. The traces of electrical signals are segmented by filtered null points and inflection points. Then, they apply a modified Prony's method to fit an exponential function to each segment within an error bound. The transition guards are estimated from the transitions' post states. Sarkar et al. propose an approach to learn a stochastic switched linear model for nonparametric systems, which is by constructing data with Hankel-like matrices and computing approximations via singular value decomposition (SVD) truncation [44]. Miriam et al. propose algorithms to apply membership-based synthesis to learn linear hybrid automata with nondeterministic guards and invariants [45]. Bernhard et al. combine abstract automata learning, model-based testing, and machine learning to learn a hybrid system, where the state space is first discretized and then a testing method is applied to generate sufficient data for the behavioral estimation in the machine-learning process [2]. Lamrani et al. propose a framework for the learning of hybrid systems [30], where candidate models are clustered from traces using feature vectors, and guard conditions are then estimated based on the segmentation of traces. However, such a framework requires a good prior knowledge of the target system to select features for the clustering.

Even though tremendous related techniques have been developed, the inference of hybrid automata is still a open and challenging problem. In this paper, we propose a framework for inferring and validating deterministic hybrid systems from another different perspective. We evaluate it with four benchmarks as well as a comparison with one state-of-the-art membership-based approach, and show that our framework can identify accurate hybrid automata given trace information and can be an effective and promising approach.

3 HYBRID AUTOMATA

Hybrid automata is a common formal modelling framework for hybrid systems that combine a finite state machine with a finite set of real-valued continuous state variables described by differential equations or inclusions. Our work mainly focuses on deterministic and synchronous models, where all constraints over state variables are specified using linear (affine) equations or inequalities. Given a set of time series traces that are generated from a hybrid system, a formal inference model, G_h, for this hybrid system is inferred as a hybrid automaton. We assume the system dynamics in each mode are characterized by an affine ODE:

$$\dot{\mathbf{x}} = A_q\mathbf{x} + B_q\mathbf{u} \tag{1}$$

where A_q and B_q are constant system matrix, and $\mathbf{x}$ indicates the state variable vector and $\mathbf{u}$ denotes an input vector. In our work, the identification problem of this continuous dynamics is the determination of A_q and B_q from traces that

are collected at a constant sampling frequency. With these traces, its discrete-time representation will be first derived and then be converted back to Equation 1. Here, its discrete-time representation is defined as

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \tag{2}$$

where matrix A and B will be used for the further mode identification and mode merging in the framework.

Definition 3.1 (Hybrid System Model). A *hybrid automaton* G_h is a tuple $G_h = \langle Q, X, f, E, \Phi, U \rangle$:

- Q is a finite set of control *modes*, and $X \subseteq \mathbb{R}^n$ is the continuous state space, an element of which is typically denoted as $\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top \in X$. The $\mathbf{x}$ is also referred as the variable vector.
- f is a vector field that describes the dynamics of X with respect to real-time, $f : Q \times X \times U \to X$. For a mode $q \in Q$, we define f as in Equation 1, where A_q and B_q are time-invariant within mode q:

$$\dot{\mathbf{x}} = f(q, \mathbf{x}, \mathbf{u}) = A_q\mathbf{x} + B_q\mathbf{u}.$$

- E denotes events or guard conditions that trigger modes switching, where $e \in E$ is an exogenous event or multiple linear inequalities (predicates) involving continuous variables. The invariant conditions are the complement.
- Φ denotes the discrete transitions: $Q \times E \to Q$. Here, $\phi : \langle q, e, q' \rangle \in \Phi$ denotes a mode transition from source mode q to a destination mode q' triggered by an event e.
- $U \subseteq \mathbb{R}^m$ denotes a continuous space of inputs, and $\mathbf{u} = [u_o; u_q]^\top \in U$ is an input consisting of an exogenous input to the system, $u_o \in \mathbb{R}^{m-1}$, and an internal constant, $u_q \in \mathbb{R}$ in each mode.

Definition 3.2 (Trajectory). A *trajectory* of G_h from a state $(q, \mathbf{x})$ to a state $(q', \mathbf{x}')$ where $q, q' \in Q$ and $\mathbf{x}, \mathbf{x}' \in X$ is a pair $\rho \triangleq (\mathfrak{Q}, \mathfrak{X})$. $\mathfrak{Q}$ and $\mathfrak{X}$ are functions that define for each time point in an interval T the mode and the values of the continuous state variables. The time points where mode switches $\phi \in \Phi$ occur are defined as timestamp changepoints $(\tau_i)_{i=0,1,\ldots,p} \in T$. The timestamp changepoints τ_i must satisfy the following conditions: (1) $\tau_0 = 0$, $\tau_i < \tau_{i+1}$ and $\tau_p = T$, (2) $\forall i \forall t \in [\tau_i, \tau_{i+1}), \mathfrak{Q}(t) = \mathfrak{Q}(\tau_i)$, (3) $\forall i \forall t \in [\tau_i, \tau_{i+1})$, the dynamics function f at each t is the same as at τ_i.

4 IDENTIFYING AND CLUSTERING DYNAMICS FROM TRACES

In this section, we present a method to estimate and cluster ordinary differential equations (ODEs) from traces. In clustering of trace segments, some works apply clustering methods from the field of machine learning [35]. However, it is challenging to select effective dynamic features to distinguish time-series data traces from different dynamics, and meanwhile the selected features may not be easily generalized to other systems. Instead, we utilize the Linear Matrix Inequality (LMI) method to detect the dynamic similarity between trace segments under a specified error tolerance.

4.1 Changepoints and Input-output Traces

The dynamics of hybrid systems is reflected in the behaviors of execution traces. One such trace is one set of a finite sequence of input signals and their corresponding outputs (or state variable values) with a constant sampling time interval t_s. From the perspective of the *trajectory* in Definition 3.2, an output trace over an execution time T is a sequence of sampled values of the continuous variables $\mathbf{x}$ in the *trajectory* over T, and it is denoted as $\mathcal{X}$. In segmenting the traces, a *changepoint* of two consecutive trace segments is characterized by an abrupt change in value or slope, which also can reflect a mode switch. Thus we use the timestamp *changepoints* $(\tau_i)_{i=0,1,\ldots,p} \in T$ from Definition 3.2 to represent the time point where mode switches happen. With such *changepoints*, an output trace $\mathcal{X} \in \mathbb{R}^{n \times l}$ with a length l will be segmented into $[\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_p]$, where $\mathcal{X}_i = \mathcal{X}_{[\tau_i, \tau_{i+1}]}$ and l_i is its length. Similarly, let $\mathcal{U} \in \mathbb{R}^{m \times l}$ be

an input trace, then $[\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_p]$ represent the input segments where $\mathcal{U}_i = \mathcal{U}_{[\tau_i, \tau_{i+1}]}$. A set $(\mathcal{X}_i, \mathcal{U}_i)$ is called a *trace segment*. This definition of traces is demonstrated by one example of state traces collected from a buck-converter system, as show in Figure 3.

In our experiments, all the traces are automatically segmented by applying the peak detection algorithm on the second-order difference of state traces. A desired trace segment should only contain a single dynamics. Despite the fact that a perfect segmentation cannot be guaranteed due to noises in traces, the clustering method based on the LMI in the following section helps filter out erroneous trace segments that contain multiple dynamics. This is because when a trace spans multiple dynamics, the solution space for its ODE estimation will have few chances to overlap with the solution space from the trace that has single dynamics. These erroneous segments can be detect by checking the number of trace segments in each cluster. Additionally, the changepoints of these mis-segmented traces do not reflect the true guard condition for mode transitions. In the estimation of the guard conditions, these changepoints will likely fall in the outlier where they become invalid. Overall, with our framework, the impact of mis-segmented traces on the inference of hybrid automata can be reduced.

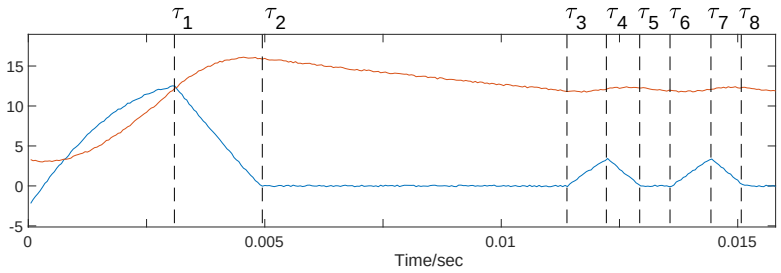

Fig. 3. One set of traces from a buck-converter system. This set of data includes the voltage and current measurement in a circuit, and here τ denotes a discontinuity, henceforth referred to as a *changepoint*.

4.2 Construction of Solution Space

Given a trace segment, A solution space is the space that contains all the possible parameter sets for Equation 2 under a pre-specified error tolerance. The error here refers to the difference between the given traces samples for learning and the traces generated from the learned dynamics. The solution space is represented by a Linear Matrix Inequality (LMI) as defined in Equation 8. Based on it, we can inspect the similarity between the dynamics of trace segments by checking whether their solution spaces overlap. The construction of the LMI starts with the dynamics formulation. To reduce the impact from noise in trace segments, we consider the dynamics in Equation 3 which is modified from Equation 2:

$$\mathbf{x}_{k+j} = A^j \mathbf{x}_k + A^{j-1} B \mathbf{u}_k + A^{j-2} B \mathbf{u}_{k+1} + \cdots + B \mathbf{u}_{k+j} \tag{3}$$

The reason is as follows. Suppose an output $x = \overline{x} + \varepsilon$ where $\overline{x}$ denotes the true value and ε denotes noise, then Equation 2 can be converted to

$$\overline{\mathbf{x}}_{k+1} + \varepsilon = A(\overline{\mathbf{x}}_k + \varepsilon) + B\mathbf{u}_k$$

but the difference of state-variables values after one time step can be so small that the noise ε may dominate the dynamics estimation, leading to an inaccurate model. To simplify the analysis in this equation and further application of LMI, the system is assumed to have a constant input $\mathbf{u} = \mathbf{u}_k$ over this j time steps. Thus, with $E = A^{j-1}B + A^{j-2}B + \cdots + B$

as one matrix, Equation 3 can be simplified to Equation 4. In this case, $\|\mathbf{x}_{k+j} - \mathbf{x}_k\|$ normally increases with j, such that a larger j can reduce the impact of the noise ϵ. For different dynamics and noise bounds, the choice of j can be determined empirically. Our experimental results of the inference of two hybrid automatons show that when j=10, accurate hybrid models can be estimated from state traces with uniformly distributed noise $[-0.05, 0.05]$.

$$\mathbf{x}_{k+j} = A^j \mathbf{x}_k + E\mathbf{u}_k \tag{4}$$

Let $\mathcal{A} = [A^j, E]$, $O = (\mathcal{X}, \mathcal{U})$ be the collected traces of the state variables and input signals of the right part of Equation 4, and $O' = \mathcal{X}$ be the trace of the state variables of the left part. Given a trace segment $(\mathcal{X}_i, \mathcal{U}_i)$ with a constant sampling interval τ_s, we can construct Equation 5 from Equation 4 that

$$O'_i - \mathcal{A}_i O_i = 0 \tag{5}$$

where $O_i = [\mathcal{X}; \mathcal{U}]_{[\tau_i, \tau_{i+1} - jt_s]} \in \mathbb{R}^{(n+m)\times(\tau_{i+1}-\tau_i-jt_s)}$, $O'_i = \mathcal{X}_{[\tau_i+jt_s, \tau_{i+1}]} \in \mathbb{R}^{n\times(\tau_{i+1}-\tau_i-jt_s)}$, and $\mathcal{A}_i = [A^j, E] \in \mathbb{R}^{n\times(n+m)}$, and τ_i, τ_{i+1} are timestamps of *changepoints*. A typical method to compute the optimal parameter sets for this trace segment is the least square method. However, empirically it will be still challenging to measure the similarity between the derived parameter matrix of different trace segments. Therefore, here we propose the solution-space approach as an alternative to handle such problems. By adding an error tolerance σ to Equation 5, we have

$$\forall h \in [1, 2, \ldots, n], \quad \frac{1}{l_i}\|\{O'_i - \mathcal{A}_i O_i\}_h\|_2 \leq \sigma \tag{6}$$

where $\{*\}_h$ denotes the error trace of the continuous variable x_h, l_i denotes the trace length, and the $\frac{1}{l_i}\|*\|$ indicates the averaged error. The function $\{*\}_h$ can be realized by right multiplying a selection matrix $C_h \in \mathbb{R}^{1\times n}$ where the hth element is 1 and the rest of elements are zero. Then Equation 6 is converted to

$$\frac{1}{l_i}\|C_h(O'_i - \mathcal{A}_i O_i)\|_2 \leq \sigma \tag{7}$$

Therefore, it is guaranteed that, for each dimension of output in all the possible models within this solution space of $\mathcal{A}$, the average error is not greater than σ. Thus, it can be guaranteed that the precision of clustered dynamics can be bounded by σ . According to Theorem 4.1, which has been proven [54], Equation 7 is equivalent to a non-strict LMI that

$$F_h = \begin{bmatrix} I & (O'_i - \mathcal{A}_i O_i)^\top C_h^\top \\ C_h(O'_i - \mathcal{A}_i O_i) & (l_i\sigma)^2 \end{bmatrix} \geq 0$$

To combine the multiple LMIs $F_1 \geq 0, F_2 \geq 0, \ldots$, they can be merged into a single LMI: $\mathbf{diag}\{F_1, F_2, \ldots\} \geq 0$, where $\mathbf{diag}\{\}$ denotes the diagonal matrix of given matrices. Therefore, considering all $(F_h)_{h\in[1,2,\ldots,n]}$, the solution space $\mathcal{S}_i$ of $\mathcal{A}_i$ for the trace segment $(\mathcal{X}_i, \mathcal{U}_i)$ can be transformed into an LMI form as Equation 8 and $F(\mathcal{A}_i) = \mathbf{diag}\{F_1, F_2, \ldots, F_n\}$.

$$\mathcal{S}_i = \{\mathcal{A}_i | F(\mathcal{A}_i) \geq 0\} \tag{8}$$

Theorem 4.1. *Suppose* $\mathbf{M}$ *is a symmetric matrix given by*

$$\mathbf{M} = \begin{bmatrix} A & B \\ B^\top & C \end{bmatrix}$$

and A *is invertible. Then, sufficient and necessary conditions for positive semidefiniteness of* ($\mathbf{M} \geq 0$) *in terms of the Schur complement are*

$$\mathbf{M} \geq 0 \Leftrightarrow A \geq 0,\ C - B^\top A^{-1} B \geq 0,\ (I - AA^{-1})B = 0$$

Algorithm 1: Identification of modes

$O \leftarrow p$ sets of sorted trace segments;
Function `FnRecursive`(O)
  $O_c, I_n \leftarrow empty$ # I_n is a set of indices ;
  if $O \neq empty$ **then**
    $\mathcal{S}_1 \leftarrow$ `FnSoluspace`(O_1) # construct a solution space;
    for $j = 2$ **to** p **do**
      $\mathcal{S}_j \leftarrow$ `FnSoluspace`(O_j) ;
      $\mathcal{S}_{temp} \leftarrow$ `FnCombine` $(\mathcal{S}_1, \mathcal{S}_j)$ # combine two solution spaces;
      $feasibility \leftarrow$ `FnInspace` $(\mathcal{S}_{temp})$ # check the feasibility of the two solution spaces;
      if $feasibility < 0$ **then**
        j add to I_n # two solution spaces are compatible and their dynamics will are clustered;
      else
        O_j add to O_c;
      end
    end
    `FnRecursive` (O_c) add to I_n # the recursive step for the unclustered trace segments;
    return I_n;
  else
    return $empty$;
  end
end

4.3 Intersection of Solution Spaces

In this section, methods for linear matrix inequalities are applied to determine the intersection between two different solution spaces. The existence of an intersection implies that the dynamics of these two segments can be clustered into one. Given two trace segments $(\mathcal{U}_1, \mathcal{X}_1)$ and $(\mathcal{U}_2, \mathcal{X}_2)$, we can construct their solution space $F(\mathcal{A}_1)$ and $F(\mathcal{A}_2)$ as LMIs according to Equation 8. To determine intersection between these two solution spaces, we can merge them into one LMI $\mathbf{diag}\{F(\mathcal{A}_1), F(\mathcal{A}_2)\} \succeq 0$ and compute its feasibility. A true feasibility indicates an intersection and that these two segments exhibit similar dynamics, and otherwise, not. The LMI approach we apply is the polynomial-time projective method proposed by Nemirovskii [37], which is one of the most efficient algorithms among interior-point methods for solving LMI problems.

The clustering method is illustrated in algorithm 1, which recursively searches for the trace segments belonging to the same cluster by checking their intersection. In the algorithm, the longer trace segment is assumed to have higher likelihood of encoding more dynamic information. Thus, trace segments are sorted in decreasing order and the longest segment is used as a reference for the rest. The function's output is the clustered index of trace segments. Here, the symbol O_j denotes the jth trace segment and the symbol $\mathcal{S}_j$ denotes its solution space in LMIs. The O_c denotes the set of trace segments that fails to be clustered with the O_1, and I_n denotes the a set of index of segments which belong to the same cluster. The functions in the algorithm are as follows. (1) Function `FnSoluspace` returns its solution space expressed in LMIs. (2) Function `FnCombine` combines two solution spaces. (3) Function `FnInspace` calculates the feasibility of two LMIs through the projective method.

For the computational complexity, let n be the total number of traces segments for the framework. The modes identification takes $O(n^2)$ operations of checking the feasibility of LMIs. After clustering, all the trace segments in the

same cluster will be applied to calculate the $\mathcal{A}$. With the sampling interval τ_s, the continuous dynamics can subsequently be derived. The clustered ODEs will be labelled with the symbol f, and then the segmented trace is converted into a ODE-label trace. Suppose a trace $\mathcal{X}$ is segmented into $[\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3]$, and $\mathcal{X}_1$, $\mathcal{X}_3$ are clustered together with a label f_1, and the $\mathcal{X}_2$ is with a label f_2, thus, we have a ODE-label trace $[f_1, f_2, f_1]$. Accordingly, we obtain two *preliminary transitions* for $\mathcal{X}_1 \rightarrow \mathcal{X}_2$ and $\mathcal{X}_2 \rightarrow \mathcal{X}_3$:

$$\langle f_1, \textit{changepoint}, f_2 \rangle, \quad \langle f_2, \textit{changepoint}, f_1 \rangle \tag{9}$$

5 INFERRING GUARD CONDITIONS

As described in Definition 3.1, there are two types of guard conditions: exogenous *event* and linear inequalities (LIs). As introduced, the model we consider is synchronous, which indicates that there is no delay between the input and output. Therefore, the *event*'s immediate impact on the system will be directly reflected in a mode switching and its corresponding changepoint in output traces, from which the association between the *event* and its mode switching can be achieved. Here, we mainly focus on the LIs' estimation.

The LI estimation is conducted for each type of preliminary transition (from (9)) that have the same source ODE and destination ODE. There generally exist various types of LIs guard conditions for transitions. Suppose two e_1 and e_2 LIs guard conditions are respectively derived for the *changepoints* in the transitions in (9), then we can update those preliminary ODE-labeled transitions to $\langle f_1, e_1, f_2 \rangle$ and $\langle f_2, e_2, f_1 \rangle$ which will be utilized for future mode merging.

Given changepoints from one type of transitions, we can estimate the LIs using affine-subspace clustering method which aims to cluster data into multiple low-dimensional planes. Here, we utilize the Random Sample Consensus (RANSAC), a statistical method proposed in [20] which is a learning technique for estimating parameters of a mathematical model by iteratively and randomly sampling a set of observed data. The observed data contain inliers, points that can be approximated by fitting to a plane, and also outliers, points that cannot be fit. The plane that are estimated from the most inliers is selected as the optimal one. In our case, the changepoints are normally close to the boundary of LIs, so RANSAC can potentially exhibit a very competitive performance compared with other methods [5]. The original RANSAC estimates one plane for one particular data group. A guard condition may consist of multiple LIs in conjunction, which means there may exist multiple planes to estimate. Inspired by the work [48, 53], we choose to apply RANSAC sequentially, to mine a new subspace from the modified data set, where the points belonging to previously found planes are removed.

The algorithm is shown in Algorithm 2. The *inData* and *inNum* denote the inlier points and their number, respectively. The *plane* denotes a candidate in one iteration while the *bestPlane*, *bestData* and *bestNum* denote information of the current best candidate. During each iteration, the function `FnRandomSample` randomly selects a candidate. The function `FnValidP` finds all its inlier points. Once a new plane is determined, the corresponding points will be removed before the next iteration. The process will be terminated after no more planes is found.

Figure 4 demonstrates one sample of LI estimation from *changepoints* of one same type of transitions. The points describe the *changepoints*, from which multiple solid lines that denotes the LIs can be estimated using the modified RANSAC. The points' positions, with respect to their lines, are used to determine the inequality sign. The logical connectivity between multiple linear inequalities is also taken into consideration. LIs are determined to be conjucted if all trace segments ahead of those *changepoints* do not satisfy them, otherwise, each LI is treated as an individual guard condition. For the changepoints that are still outliers after the algorithm termination, we use a label 0 to indicate an invalid estimation. The transition with such guard condition will be removed from the further automata inference.

Algorithm 2: Identification of Multi-planes

```
while true do
    bestPlane, bestData ← empty ;
    bestNum ← 0;
    Remove(bestData);
    for n = 1 to η do
        plane ← FnRandomSample(data) # randomly create a candidate;
        inNum, inData ← FnValidP(plane) # determine the inliers and outliers;
        if inNum > γ & inNum > bestNum then
            bestNum, bestData ← inNum, inData # update the current optimal candidate;
            bestPlane ← plane;
        end
    end
    if bestPlane ≠ empty then
        Output (bestPlane);
    else
        break
    end
end
```

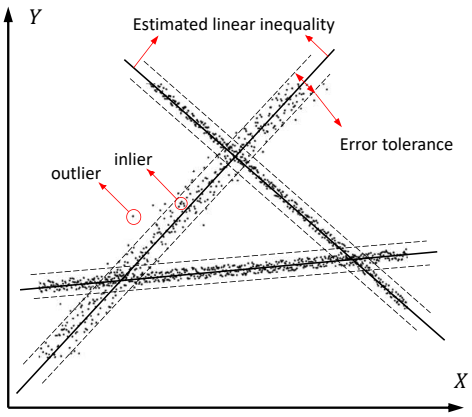

Fig. 4. Linear inequality estimation using RANSAC

Similar to the original RANSAC, the modified one has three parameters to specify: (1) error tolerance, λ, to check a point's compatibility with a model candidate, (2) the iteration number, η, for each model estimation, and (3) the threshold number of compatible points, γ, that indicates a valid estimation. In our work, the error tolerance, λ, is described by the distance between a point and its corresponding affine hyperplane. There do not exist straightforward methods to determine these three parameters, but we are able to approximate them by decreasing it from a large value. Since all the changepoints are near the boundary of their planes, there should be an error tolerance λ, such that most of the points become inliers. Decreasing, iteratively, from a large value candidate to approach such a λ helps achieve a robust estimation. For the threshold number, suppose that the probability of one point being compatible with all the

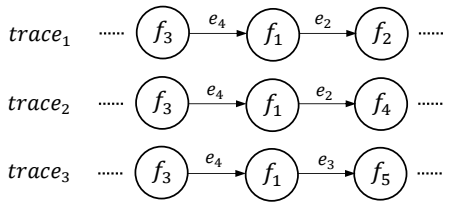

Fig. 5. Illustration of mode merging. According to the compatibility criterion, f_3 and f_1 in $trace_1$ are respectively compatible with the f_3 and the f_1 in $trace_3$. However, they are not compatible with the states in $trace_2$ because in the subsequent transition, the guard condition e_2 triggers different transitions from the f_1 state.

planes is equal, then, the selected γ should not exceed n/m, where n is the total number of data points and m is the number of linear inequalities involved in that transition condition. The iteration number η can be approximated with the method in [20], which is based on the assumption of the probability of only inliers being selected in some iterations and the probability of one single inlier being selected each time.

6 MERGING MODES

In this section, we introduce the concept of the prefix tree acceptor (PTA) to help merge the ODE-label traces from the previous section and construct the final hybrid model. Each ODE which represents a type of dynamics will be regarded as a candidate mode. There exist many heuristic algorithms focusing on inferring automata grammar from a set of labeled strings. To our best knowledge, the element in the string only represent input events or internal guard conditions that trigger mode switches, without considering the dynamics in modes. In our proposed framework, besides the estimated guard condition of each transition, the dynamics in each mode has been classified before the merging, which can be used for a more robust inference. As shown in Figure 5, each subtree in a PTA represents one processed mode trace. Each mode is characterized by their own ODE, f, and the guard condition in each mode switching is denoted by e. Unlike the work [14, 39] which shows that the similarity of two modes is associated with the probability of staying in or transitioning out of a mode, our method evaluates the compatibility of two modes using their derived ODEs and transition conditions. To avoid creating a non-deterministic system during the process of merging, modes are *compatible* under the following two conditions:

- First, the source ODE, destination ODE and guard conditions of two transitions are the same, and meanwhile, in the subsequent transition where the destination ODE is the next source ODE, there do not exist different transitions that are triggered by the same guard condition. Then the modes involved in these two transition are compatible. This situation is illustrated in Figure 5.
- Second, the first segments in each trace have the same dynamics and represent the same initial mode.

In the algorithm, merging of modes is associated with the merging of mode transitions. We first construct a 6-tuple $\langle label_1, e, label_2, id_1, id_2, times \rangle$ for each mode transitions in traces. $label_1$ and $label_2$ respectively denotes the labels of the source and destination ODE. Item e denotes the guard condition. Modes with the same ODE are not necessarily compatible, which will be demonstrated by the *navgation* system in the case study. Some modes exhibiting the same dynamics are separated due to different guard conditions. For a clear identification, the modes in all the preliminary traces will also be assigned with a unique index id in addition to the ODE label and this index is used to represent the

unicity of a mode. The id_1 and id_2 respectively denote the index of the source and destination. Item *times* is used to count the number of transitions that are merged to the current one.

During the merging process, the 6-tuples are checked and merged according to the compatibility criterion. The algorithm is shown in Algorithm 3. For each pair of tuples *ituple* and *jtuple*, we check their compatibility by the function `FnCompatibility` which refers to the compatibility criterion. If they are compatible, *jtuple* will be merged to *ituple*, and id_1, id_2 of *jtuple* are modified to be consistent with *ituple*'s. Meanwhile, the function `FnModifyTuple` is applied to search the rest of transitions and modify the modes having the same indices as *jtuple*'s. Thus the connectivity between transitions can be maintained. Afterwards, the *jtuple* is emptied and the *times* in *ituple* is increased by one. For the analysis of the algorithm complexity, let n be the number of *tuple*. Then the merging modes takes $O(n^2)$ operations of checking the compatibility of *tuple*s.

Algorithm 3: Merging Modes

```
for i = 1 to num_tuple do
    ituple ← tuples[i];
    if ituple ≠ empty then
        for j = i + 1 to num_tuple do
            jtuple ← tuples[j];
            if FnCompatibility(ituple, jtuple) then
                FnModifyTuple(ituple, jtuple);
                tuples[j] ← empty ;
                ituple.times + +;
            end
        end
    end
end
return tuples;
```

6.1 Parameter Selection

Multiple parameters need to be set in the identification of dynamics and the inference of guard conditions, and their selection can determine the performance of the framework. The identification of dynamics includes two parameters. One is the parameter j in Equation 3 which denotes the number of steps and is utilized to reduce the impact of the noise. Overlarge values of j result in less impact of the noise. The other one is the error tolerance ε which is for the solution space of ODE parameters. An overlarge value of ε generates a larger solution space, which may result in clustering together trace segments with different dynamics. Too small a value leads to a smaller solution space, which may result in classifying segments with similar dynamics into different clusters. In the guard-conditions, the λ determines changepoints' compatibility with a mode candidate. An overlarge value can misclassify changepoints into a different LI. While too small a λ will generate more outliers and thus lose more information. For the η, a large number will increase the robustness of the LI estimation but will also increase the computational burden. The threshold number γ is the number of changepoints needed to validate an estimation. An overlarge value can cause estimation failure while too small a value can yield a large amount of LIs and thus cause overfitting issues.

7 CASE STUDIES AND EVALUATION

In this section, we study several case studies to evaluate the proposed hybrid automata learning framework on different systems, following the validation overview from Figure 2. The chosen systems are a navigation system, a multi-room heating system [18], and a DC-DC buck converter system [11, 38]. They all have linear ODEs, guards, and invariants. We also compare our method with a membership-based algorithm [45] on a simplified heating system. They are designed as Simulink/Stateflow models that can generate training traces for the hybrid automata inference framework and testing traces for validation of our methodology. The methodology is implemented in a prototype software tool in Matlab, building on the HyST software tool [7] and its integration with Simulink/Stateflow [6]. [1]. In the evaluation, we utilize two methods to measure the proximity between the inferred system and the original. The first one is comparing their reachable states given an input set. Using the learned hybrid automata that are generated from our framework, we compute their reachable sets using SpaceEx to compare the original model to the learned model as a form of equivalence checking. The second evaluation is the conformance degree proposed in [1, 12]. As defined, it can provide a proximity measure between two output traces in both space and time (see Table 1). It is noteworthy that the component τ of the conformance degree relies on the running time T of traces. A longer trace will lead to a larger value τ. This is because the estimation error in the model-transition condition as well as in the dynamics equations will be reflected on the difference of transition time between the original system and the inferred one. Such difference can be accumulated with each transition. This is also a challenging issue, and to our best knowledge, there are few effective solutions. Overall, the τ that represents the worst transition-time difference is generally related to the last mode transition that occurs in the trace. And a large value of τ can be due to many mode transitions

Definition 7.1 (Conformance Degree). Given output traces for time $T \in \mathbb{R}_{>0}$, a maximum number of mode switches $J \in \mathbb{N}$, and parameters $\tau_c, \epsilon > 0$, two traces y_1 and y_2 are (T, J, τ_c, ϵ)-close, if (1) for all $(t_1, j_1) \in y_1$ such that $t_1 < T$ and $j_1 < J$, there exists $(t_2, j_2) \in y_2$ such that $|t_1 - t_2| \leq t_c$ and $\|y_1(t_1, j_1) - y_2(t_2, j_2)\| \leq \epsilon$, (2) for all $(t_2, j_2) \in y_2$ such that $t_2 < T$ and $j_2 < J$, there exists $(t_1, j_1) \in y_1$ such that $|t_1 - t_2| \leq t_c$ and $\|y_1(t_1, j_1) - y_2(t_2, j_2)\| \leq \epsilon$.

Table 1. Execution Time and accuracy evaluation.

Case Study	**Total Time (sec)**	**Trace Segments**	**Conformance Degree** $(T(sec), J, \tau(sec), \epsilon)$
Navigation System	11157	299	(1.8, 4, 0.21, 0.22)
Multi-room Heating System	8866	691	(40, 8, 1.2, 0.98)
Buck Converter	65	144	(0.02, 13, 0.0015, 0.17)
Cooperative Vehicles	54	80	(10, 2, 0.0000, 0.1265)

7.1 Navigation System

This system deals with dynamics of an object in the $\mathbb{R}^2$ plane with $m \times n$ grids. In each grid, the desired velocity along the x and y axes are respectively set to $\sin(i * \pi/4)$ and $\cos(i * \pi/4)$, where $i = 0, 1, \ldots, 7$, and the length and width of each are set to 1. The system to be learned is shown in Figure 6. Given the desired velocity $\mathbf{v}_d$, the dynamics of the actual velocity $\mathbf{v}$ is described by the differential equation $\dot{\mathbf{v}} = A(\mathbf{v} - \mathbf{v}_d)$ where $A = [-1.2,\ 0.1;\ 0.1, -1.2]$.

[1]Code for the prototype HAutLearn tool and examples is available online at: https://github.com/verivital/hautlearn

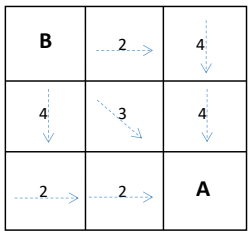

Fig. 6. Navigation system in a 3×3 grid. The label i refers to the parameter to calculated the desired velocity. The object needs to reach the grid labelled **A** and meanwhile avoid the grid labelled **B**.

The object's start position can be in any grid except for **A** and **B**. Here, we choose to learn one hybrid automaton for each starting grid. By fixing the initial position in one specified grid and trying different initial positions and velocities, we can generate sufficient trajectory traces which are respectively, position in X direction, position in Y direction, velocity in X direction and velocity in Y direction. Then a hybrid automaton is estimated through our framework to approximate their dynamics. We estimated one hybrid automaton for the traces starting from the bottom left grid. For the learning, we set $\sigma = 10^{-4}$ for clustering the trace segments, $\varepsilon = 0.01$, $\eta = 10^3$ and $\gamma = 10$ for estimating the LIs. Here, 81 traces are collected with a sampling time $t_s = 0.01s$. The estimated state transitions is listed in Table 2, and the details of labels e and f is in Table 6 in Appendix A.1. Accordingly, a hybrid automaton is constructed as shown in Figure 7. The comparison of reachable states with the original system and accuracy evaluation are shown in Figure 8 and Table 1.

$label_1(f)$	$guard(e)$	$label_2(f)$	id_1	id_2	$times$
2	1	1	1	2	66
1	3	3	2	3	54
3	5	2	3	1	12
2	6	0	1	5	38
1	7	2	2	1	14
3	4	2	3	4	29
2	2	1	4	6	29
1	8	0	6	7	42
3	9	1	3	6	13

Table 2. Inferred transitions of the navigation system.

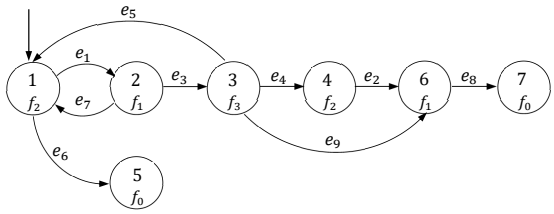

Fig. 7. Inferred hybrid automaton for the Navigation system. The number indicates each unique mode. Symbol f indicates their dynamics. Symbol e indicates the guard conditions

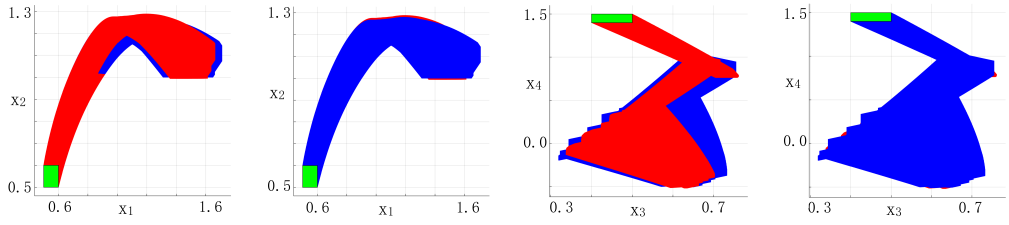

Fig. 8. Navigation: Red domain is reachable states of the inferred system while the blue domain is the reachable area of the original system. The green domain is the initial states, which is $[0.5 \leq x_1 \leq 0.6 \wedge 0.5 \leq x_2 \leq 0.6 \wedge 1.4 \leq x_3 \leq 1.5 \wedge 1.4 \leq x_4 \leq 1.5]$.

7.2 Buck Converter

A buck converter is a DC-DC power converter that steps down voltage from its input to its output. It exhibits both continuous and discrete behaviors because of the presence of passive elements and switching components. Here we consider a closed-loop DC-DC buck converter in [11]. It takes a DC voltage at its input V_{in} and then adjust its output v_c according to the V_{ref} by controlling the operation of the MOSFET switch. It is a time-independent hybrid system where there are two state variables, voltage across capacitor v_c and current through the inductor I. The switching conditions of its controller include an upper switching boundary $V_{ref} + \delta$ and a lower switching boundary $V_{ref} - \delta$.

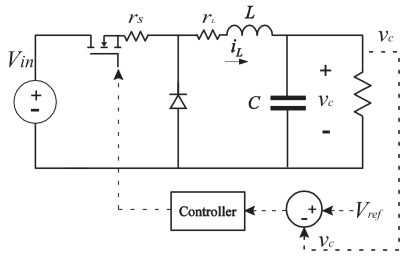

Fig. 9. Circuit of a closed-loop buck converter

The state traces are collected using Simulink where an uniformly distributed noise with an range $[-0.05, 0.05]$ was added. The sampling time t_s was set to 5×10^{-5} seconds and the total running time for each execution was 0.02 seconds. The range of the initial states are set to $I = [0, 30]$ and $v_c = [0, 15]$ from which 11 traces were collected. For the learning, we set $\sigma = 2 \times 10^{-2}$ for segments clustering, and $\varepsilon = 0.04$, $\eta = 10^3$ and $\gamma = 10$ for estimating the LIs. The inferred hybrid automaton is shown in Figure 10 and the dynamics information is shown in Table 7 in the Appendix A.2. The comparison of reachable states and accuracy evaluation are shown in Figure 11 and Table 1.

$label_1(f)$	$guard(e)$	$label_2(f)$	id_1	id_2	$times$
2	1	3	1	2	17
1	3	2	3	1	32
3	2	1	2	3	11

Table 3. Inferred mode transitions for the buck converter.

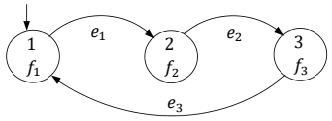

Fig. 10. Inferred hybrid automaton of the buck converter

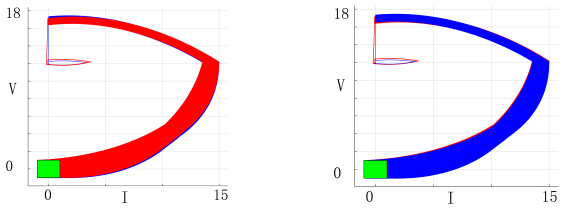

Fig. 11. Buck Converter: red domain denotes reachable states of the inferred system while the blue domain denotes the reachable area of the original system. The green is the range of initial states which is $[-1 \leq I \leq 1 \wedge -1 \leq v_c \leq 1]$.

7.3 The Multi-room Heating System

The system includes multiple rooms. The temperature in each room is controlled by one heater and depends on the outside temperature as well as the temperature in the adjacent rooms. Let x_i denote the temperature in room i and u denote the outside temperature. The temperature of each room exhibits linear dynamics with the heaters' power status, the difference between the room's temperature and the outside temperature, and other rooms, which is described by

$$f: \dot{x}_i = c_i h_i + b_i(u - x_i) + \sum_{j \neq i} a_{i,j}(x_j - x_i), \tag{10}$$

where $a_{i,j}$, b_j, c_i are constant and $h_i \in \{0, 1\}$ denotes the heater's status. $h_i = 0$ indicates the heater is not in room i or the heater is off. The heater in room i is on if $x_i \leq on_i$ and off $x_i \geq off_i$. A heater will move to room i from room j if all of the following conditions hold: (1) no heaters in room i, (2) one heater in room j, (3) $x_i \leq get_i$, (4) $x_j - x_i \geq dif_i$.

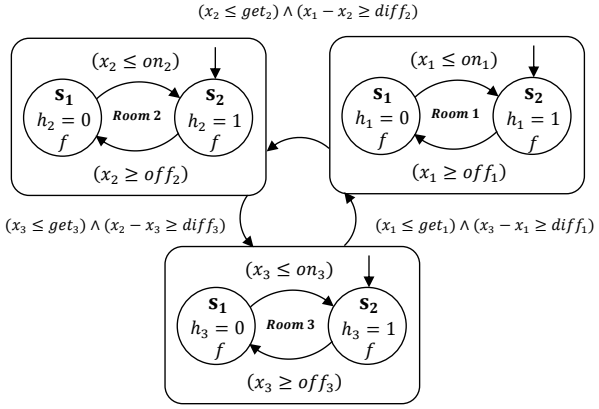

Fig. 12. Hybrid automaton model of the multi-room heating system.

For our experiment, the heating system is set to have three rooms and one heater. Since there may be multiple transition conditions holding simultaneously and the system may become non-deterministic, we restrict that there is only one destination room for each source room. Then, the system can be modelled as shown in Figure 12, which has $\mathbf{u}$ and $[x_1, x_2, x_3]$ as the input and output, respectively. The input/output traces are collected by running simulations in Matlab with a sampling interval of 0.1 seconds. For the learning, we set $\sigma = 5 \times 10^{-5}$ for segments clustering, and $\varepsilon = 0.05$, $\eta = 10^3$ and $\gamma = 10$ for estimating the LIs. The state transitions in 6-tuples generated from the framework is shown in Table 4. The final hybrid automata is shown in Figure 13, which has 6 discrete mode, 4 distinct ODEs, and 9 mode switches in total. The initial mode is the mode 1. The parameters of the guard conditions e and ODEs f are shown in Table 8 in Appendix A.3. The comparison of reachable states is shown in Figure 14. These reachable domains are approximated by simulating 1000 traces because SpaceEx does not support the non-convex linear constraints. Its accuracy evaluation is shown in Table 1.

7.4 Cooperative Vehicles

This benchmark is a platoon of three three autonomously-driven vehicles following a leader [34], as shown in Figure 15. The difference between the distance d_i of the vehicle i to its predecessor and a reference distance $d_{ref,i}$ is defined as the

$label_1(f)$	$guard(e)$	$label_2(f)$	id_1	id_2	$times$
2	2	1	1	2	107
1	3	3	2	3	103
3	4	2	3	1	110
2	5	4	1	4	24
4	6	2	4	1	24
1	7	4	2	5	98
4	8	1	5	2	95
3	9	4	3	6	52
4	10	3	6	3	52

Table 4. Inferred transitions for the multi-room heating system.

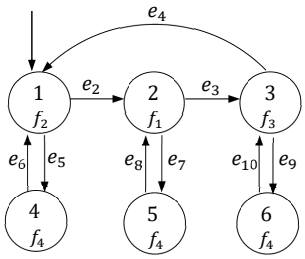

Fig. 13. Inferred hybrid automaton of the heating system.

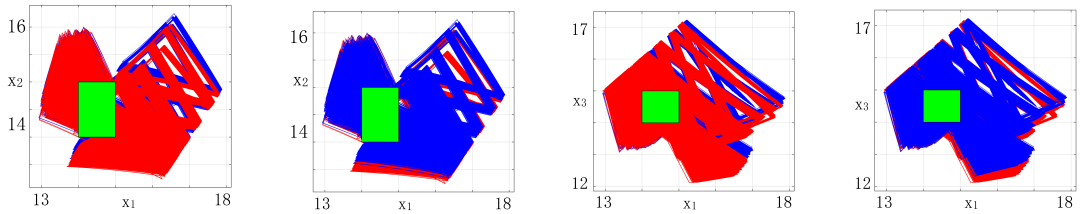

Fig. 14. Heater: Red domain denotes reachable states of the inferred system while the blue domain denotes the reachable area of the original system. The green denotes the initial states that is $[14 \leq x_1 \leq 15 \wedge 14 \leq x_2 \leq 15 \wedge 14 \leq x_3 \leq 15]$.

space error e_i. The dynamics of the platoon is as follows:

$$\dot{x} = Ax + Ba_L, \tag{11}$$

where the state vector x consists of 9 variables and $x = [e_1, \dot{e_1}, a_1, e_2, \dot{e_2}, a_2, e_3, \dot{e_3}, a_3]$ with a_i being the acceleration of vehicle i, A and B are constant system matrix, and a_L denotes the acceleration of the leader vehicle. In the case of radio communication, A and B are given as follows:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1.60 & 4.86 & -3.57 & -0.81 & 0.42 & -0.04 & -0.19 & 0.36 & -0.09 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0.87 & 3.81 & -0.07 & 1.19 & 3.62 & -3.23 & -0.59 & 0.12 & -0.07 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0.71 & 3.57 & -0.09 & 0.84 & 3.25 & -0.08 & 1.27 & 3.07 & -3.13 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

While in the case of no communication, A and B are given as follows:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1.60 & 4.86 & -3.57 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.19 & 3.62 & -3.23 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0.71 & 3.57 & -0.09 & 0.84 & 3.25 & -0.08 & 1.27 & 3.07 & -3.13 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The platoon is modelled as a hybrid automaton. Every two seconds, the radio communication breaks down for the following two seconds. The breakdowns trigger discrete switches, where a timer is reset. For the training data, we collect 16 traces using Simulink with the running time and sampling time, respectively, set to 10 seconds and 0.05 seconds. The collected traces are also added with uniformly distributed noise $[-0.05, 0.05]$. For the learning, we set $\sigma = 8 \times 10^{-3}$ for segment clustering, and $\varepsilon = 0.01$, $\eta = 10^5$, and $\gamma = 10$ for estimating the LIs. The state transitions in 6-tuples generated from the framework is shown in Table 5. The final hybrid automaton is shown in Figure 16, which has 2 discrete modes, 2 distinct ODEs, and 2 mode switches in total. The initial mode is the mode 1. The parameters of the guard conditions e and ODEs f are shown in Table 9 in Appendix A.4. The comparison of reachable states is shown in Figure 14. The accuracy evaluation is shown in Table 1

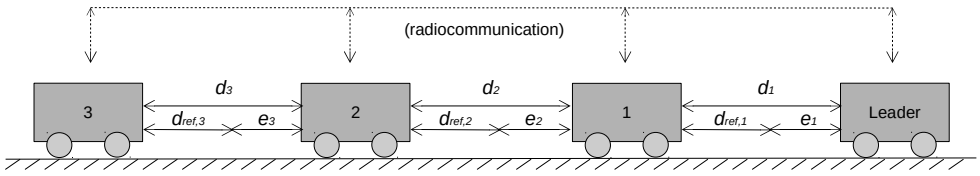

Fig. 15. Cooperative platoon of three vehicles and a leader vehicle

$label_1(f)$	$guard(e)$	$label_2(f)$	id_1	id_2	$times$
2	1	3	2	1	20
3	2	2	1	2	15

Table 5. Estimated hybrid automaton for the cooperative vehicles. The transition in red with $label_2$ being 0 is an erroneous transition

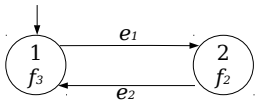

Fig. 16. Inferred hybrid automaton of the cooperative vehicles

7.5 Comparison on A Heating System

We compare our framework with a membership-based algorithm (Hysynth) for learning linear hybrid system from traces [45]. This methods defines the continuous dynamics of models with constant different equations which generally suffices to estimate an arbitrary continuous function. As claimed, this algorithm can learn an automaton with nondeterministic guard conditions and invariants with piecewise linear functions that are derived from input-output traces. We are not able to fit multi-variable traces using their source code, although there is no such limitation claimed in their work. Therefore, we choose to evaluate our framework against this algorithm in a simple case study. The target hybrid system is a heating system with one heater controlling the temperature x, which is modelled as shown in Figure 17. For the learning, 5 traces are collected with a time horizon of 20 seconds, a sampling interval 0.1 second, and different initial conditions. The piecewise linear functions are created for the membership-based method with an error bound $\epsilon = 0.1$.

Our framework can successfully learn an automaton with similar dynamics as shown in Figure 18. The running time is 5 seconds. While the membership-based method derives an automaton with 40 discrete modes and 71 mode switches, where 6 of the discrete modes are unreachable. Its running time is 2 seconds. Trace samples generated from the learned systems are shown in Figure 19. We can notice that the system learned with Hysynth terminates early and yields incomplete traces that are in red. This is mainly because the dynamics violates the invariant of a discrete mode

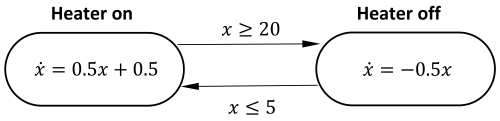

Fig. 17. Heating system with one heater

that do not have successor discrete modes to switch to. So, this inferred automaton fails to recover behaviors of the original system. The reason of this undesirable performance may be because it can deal with simple dynamics of the form $\dot{x} + c = 0$, but cannot be applied to the more general ODEs allowed by our method.

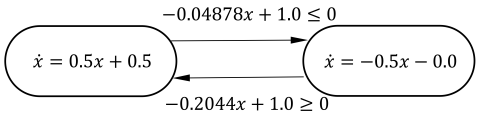

Fig. 18. Inferred hybrid automaton from the heating system with one heater

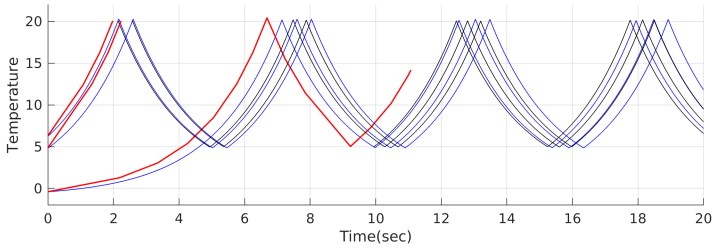

Fig. 19. Comparison of hybrid automatons learned by our method and the membership-based method. The black traces are from the original system, the blue traces are from the system learned by our method. While the red traces are from the system learned by the membership-based method.

8 CONCLUSION

This paper presents a framework to mine and learn hybrid automata with linear (affine) constraints and ODEs from input and output traces. It first clusters and estimates ODEs for the segmented traces by checking the intersection of their solution space using an LMI method. The obtained ODEs for segments defined by discontinuities in the traces are learned as potential discrete modes. Subsequently, a modified subspace-clustering method is applied to estimate the linear inequalities that describe the transition guard conditions from the collected changepoints. With the potential modes and classified events, a PTA method is applied to merge the achieved states and generate the hybrid automaton. The utility of this framework is validated by comparing approximated traces with the source traces from which the automaton is learned. There are multiple directions to improve our framework. In future work, we plan to explore improvements in the capability of data preprocessing, such as noise filtering, so that it can have better scalability. As discussed, a robust method of trace segmentation is essential for the inference of hybrid automaton, and further research

is needed in that direction. Another potential enhancement is extending this framework to nonlinear hybrid systems by exploring methods to estimate the nonlinear dynamics of each trace segment. Further case studies using black-box models and runtime monitoring can be conducted, but likely will depend on improving scalability as just discussed.

REFERENCES

[1] Houssam Abbas, Hans Mittelmann, and Georgios Fainekos. 2014. Formal property verification in a conformance testing framework. In *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE)*. IEEE, 155–164.

[2] Bernhard K Aichernig, Roderick Bloem, Masoud Ebrahimi, Martin Horn, Franz Pernkopf, Wolfgang Roth, Astrid Rupp, Martin Tappler, and Markus Tranninger. 2019. Learning a Behavior Model of Hybrid Systems Through Combining Model-Based Testing and Machine Learning. In *IFIP International Conference on Testing Software and Systems*. Springer, 3–21.

[3] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. 1995. The algorithmic analysis of hybrid systems. *Theoretical computer science* 138, 1 (1995), 3–34.

[4] Dana Angluin. 1987. Learning regular sets from queries and counterexamples. *Information and computation* 75, 2 (1987), 87–106.

[5] Ery Arias-Castro and Jue Wang. 2017. RANSAC Algorithms for Subspace Recovery and Subspace Clustering. (2017). arXiv preprint arXiv:1711.11220.

[6] Stanley Bak, Omar Ali Beg, Sergiy Bogomolov, Taylor T. Johnson, Luan Viet Nguyen, and Christian Schilling. 2017. Hybrid automata: from verification to implementation. *Software Tools for Technology Transfer (STTT)* (Aug. 2017).

[7] Stanley Bak, Sergiy Bogomolov, and Taylor T. Johnson. 2015. HyST: A Source Transformation and Translation Tool for Hybrid Automaton Models. In *Proc. of the 18th Intl. Conf. on Hybrid Systems: Computation and Control (HSCC)*. ACM.

[8] Laurent Bako. 2011. Identification of switched linear systems via sparse optimization. *Automatica* 47, 4 (2011), 668–677.

[9] Laurent Bako and René Vidal. 2008. Algebraic identification of MIMO SARX models. In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 43–57.

[10] RS Baptista, JY Ishihara, and GA Borges. 2011. Split and merge algorithm for identification of piecewise affine systems. In *Proceedings of the 2011 American Control Conference*. IEEE, 2018–2023.

[11] Omar Ali Beg, Houssam Abbas, Taylor T Johnson, and Ali Davoudi. 2017. Model Validation of PWM DC–DC Converters. *IEEE Transactions on Industrial Electronics* 64, 9 (2017), 7049–7059.

[12] Omar Ali Beg, Houssam Abbas, Taylor T. Johnson, and Ali Davoudi. 2017. Model Validation of PWM DC-DC Converters. *IEEE Transactions on Industrial Electronics. (TIE)* (June 2017).

[13] Khaled Boukharouba, Laurent Bako, and S Lecoeuche. 2009. Identification of piecewise affine systems based on dempster-shafer theory. *IFAC Proceedings Volumes* 42, 10 (2009), 1662–1667.

[14] Rafael C Carrasco and Jose Oncina. 1999. Learning deterministic regular grammars from stochastic samples in polynomial time. *RAIRO-Theoretical Informatics and Applications* 33, 1 (1999), 1–19.

[15] David R. Cok and Scott C. Johnson. 2014. SPEEDY: An Eclipse-based IDE for invariant inference. In *F-IDE*. 44–57. https://doi.org/10.4204/EPTCS.149.5

[16] M.D. Ernst, J. Cockrell, William G. Griswold, and D. Notkin. 2001. Dynamically discovering likely program invariants to support program evolution. *Software Engineering, IEEE Transactions on* 27, 2 (2001), 99–123. https://doi.org/10.1109/32.908957

[17] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. 2007. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming* 69, 1–3 (Dec. 2007), 35–45.

[18] Ansgar Fehnker and Franjo Ivančić. 2004. Benchmarks for Hybrid Systems Verification. In *Hybrid Systems: Computation and Control*, Rajeev Alur and George J. Pappas (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 326–341.

[19] Giancarlo Ferrari-Trecate, Marco Muselli, Diego Liberati, and Manfred Morari. 2003. A clustering technique for the identification of piecewise affine systems. *Automatica* 39, 2 (2003), 205–217.

[20] Martin A Fischler and Robert C Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24, 6 (1981), 381–395.

[21] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *Computer Aided Verification (CAV) (LNCS)*. Springer.

[22] Andrea Garulli, Simone Paoletti, and Antonio Vicino. 2012. A survey on switched and piecewise affine system identification. *IFAC Proceedings Volumes* 45, 16 (2012), 344–355.

[23] ME Gegundez, Javier Aroba, and José Manuel Bravo. 2008. Identification of piecewise affine systems by means of fuzzy clustering and competitive learning. *Engineering Applications of Artificial Intelligence* 21, 8 (2008), 1321–1329.

[24] Radu Grosu, Sayan Mitra, Pei Ye, Emilia Entcheva, IV Ramakrishnan, and Scott A Smolka. 2007. Learning cycle-linear hybrid automata for excitable cells. In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 245–258.

[25] Thomas A Henzinger. 2000. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*. Springer, 265–292.

[26] Thomas A Henzinger and Peter W Kopke. 1999. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science* 221, 1-2 (1999), 369–392.

[27] Anca Maria Ivanescu, Thivaharan Albin, Dirk Abel, and Thomas Seidl. 2011. Employing correlation clustering for the identification of piecewise affine models. In *Proceedings of the 2011 workshop on Knowledge discovery, modeling and simulation*. 7–14.

[28] Xing Jin and Biao Huang. 2010. Robust identification of piecewise/switching autoregressive exogenous process. *AIChE journal* 56, 7 (2010), 1829–1844.

[29] Chow Yin Lai, Cheng Xiang, and Tong Heng Lee. 2010. Identification and control of nonlinear systems via piecewise affine approximation. In *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 6395–6402.

[30] Imane Lamrani, Ayan Banerjee, and Sandeep KS Gupta. 2018. HyMn: mining linear hybrid automata from input output traces of cyber-physical systems. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE, 264–269.

[31] Lennart Ljung. 2010. Perspectives on system identification. *Annual Reviews in Control* 34, 1 (2010), 1–12.

[32] Lenhart Ljung and Svante Gunnarsson. 1990. Adaptation and tracking in system identification—a survey. *Automatica* 26, 1 (1990), 7–21. https://doi.org/10.1016/0005-1098(90)90154-A

[33] Yi Ma and René Vidal. 2005. Identification of deterministic switched ARX systems via identification of algebraic varieties. In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 449–465.

[34] Jan P Maschuw, Günter C Keßler, and Dirk Abel. 2008. LMI-based control of vehicle platoons for robust longitudinal guidance. *IFAC Proceedings Volumes* 41, 2 (2008), 12111–12116.

[35] Ramy Medhat, S Ramesh, Borzoo Bonakdarpour, and Sebastian Fischmeister. 2015. A framework for mining hybrid automata from input/output traces. In *Proceedings of the 12th International Conference on Embedded Software*. IEEE Press, 177–186.

[36] Koji Mikami, Hiroyuki Okuda, Shun Taguchi, Yuichi Tazaki, and Tatsuya Suzuki. 2010. Model predictive assisting control of vehicle following task based on driver model. In *2010 IEEE International Conference on Control Applications*. IEEE, 890–895.

[37] Arkadii Nemirovskii and Pascal Gahinet. 1994. The projective method for solving linear matrix inequalities. In *American Control Conference, 1994*, Vol. 1. IEEE, 840–844.

[38] Luan Viet Nguyen and Taylor T. Johnson. 2014. Benchmark: DC-to-DC Switched-Mode Power Converters (Buck Converters, Boost Converters, and Buck-Boost Converters). In *Applied Verification for Continuous and Hybrid Systems Workshop (ARCH 2014)*. Berlin, Germany.

[39] Oliver Niggemann, Benno Stein, Asmir Vodencarevic, Alexander Maier, and Hans Kleine Büning. 2012. Learning Behavior Models for Hybrid Timed Systems.. In *AAAI*, Vol. 2. 1083–1090.

[40] Henrik Ohlsson and Lennart Ljung. 2011. Identification of piecewise affine systems using sum-of-norms regularization. *IFAC Proceedings Volumes* 44, 1 (2011), 6640–6645.

[41] Simone Paoletti, Aleksandar Lj Juloski, Giancarlo Ferrari-Trecate, and René Vidal. 2007. Identification of hybrid systems a tutorial. *European journal of control* 13, 2-3 (2007), 242–260.

[42] Andrea Pferscher, Bernhard Aichernig, and Martin Tappler. 2020. From Passive to Active: Learning Timed Automata Efficiently. In *12th NASA Formal Methods Symposium*.

[43] Harald Raffelt, Bernhard Steffen, and Therese Berg. 2005. Learnlib: A library for automata learning and experimentation. In *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*. ACM, 62–71.

[44] Tuhin Sarkar, Alexander Rakhlin, and Munther A Dahleh. 2019. Nonparametric System identification of Stochastic Switched Linear Systems. *arXiv preprint arXiv:1909.04617* (2019).

[45] Miriam García Soto, Thomas A Henzinger, Christian Schilling, and Luka Zeleznik. 2019. Membership-Based Synthesis of Linear Hybrid Automata. In *International Conference on Computer Aided Verification*. Springer, 297–314.

[46] Adam Summerville, Joseph Osborn, and Michael Mateas. 2017. Charda: Causal hybrid automata recovery via dynamic analysis. *arXiv preprint arXiv:1707.03336* (2017).

[47] Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. 2019. Time to Learn – Learning Timed Automata from Tests. In *Formal Modeling and Analysis of Timed Systems*, Étienne André and Mariëlle Stoelinga (Eds.). Springer International Publishing, 216–235.

[48] Philip HS Torr. 1998. Geometric motion segmentation and model selection. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 356, 1740 (1998), 1321–1340.

[49] Sicco Ewout Verwer. 2010. *Efficient identification of timed automata: Theory and practice*. Ph.D. Dissertation.

[50] René Vidal. 2004. Identification of PWARX hybrid models with unknown and possibly different orders. In *Proceedings of the 2004 American Control Conference*, Vol. 1. IEEE, 547–552.

[51] René Vidal. 2008. Recursive identification of switched ARX systems. *Automatica* 44, 9 (2008), 2274–2287.

[52] René Vidal, Stefano Soatto, Yi Ma, and Shankar Sastry. 2003. An algebraic geometric approach to the identification of a class of linear hybrid systems. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, Vol. 1. IEEE, 167–172.

[53] Etienne Vincent and Robert Laganiére. 2001. Detecting planar homographies in an image pair. In *Image and Signal Processing and Analysis, 2001. ISPA 2001. Proceedings of the 2nd International Symposium on*. IEEE, 182–187.

[54] Fuzhen Zhang. 2006. *The Schur complement and its applications*. Vol. 4. Springer Science & Business Media.

A APPENDIX: ADDITIONAL DETAILS FOR THE CASE STUDIES

A.1 Navigation System

Table 6. Guard conditions e and Dynamics f of the learned automata for the navigation system.

e	e_1	$0.0049x - 0.9980y - 0.0134v_x + 0.0217v_y + 1 \leq 0$
	e_2	$-0.4904x - 0.0049y + 0.0030v_y + 1 \leq 0$
	e_3	$-1.0065x + 0.0031y + 0.0201v_x - 0.0028v_y + 1 \leq 0$
	e_4	$-0.0045x - 0.4971y + 0.0011v_x + 0.0064v_y + 1 \leq 0$
	e_5	$0.0083x - 1.0007y - 0.0166v_x + 0.0139v_y + 1 \geq 0$
	e_6	$-0.5018x + 0.0067v_x - 0.0030v_y + 1 \leq 0$
	e_7	$0.0011x - 1.0070y + 0.0159v_x + 0.0126v_y + 1 \geq 0$
	e_8	$0.0054x - 0.9710y - 0.0176v_x + 0.0521v_y + 1 \geq 0$
	e_9	$-0.4784x + 0.0019y - 0.0449v_x + 0.0168v_y + 1 \leq 0$
f	f_1	$A = \begin{bmatrix} 0.0000 & 0.0000 & 0.1000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.1000 \\ 0.0000 & 0.0000 & -0.1200 & 0.0100 \\ 0.0000 & 0.0000 & 0.0100 & -0.1200 \end{bmatrix}, B = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.0100 \\ -0.1200 \end{bmatrix}$
	f_2	$A = \begin{bmatrix} 0.0000 & 0.0000 & 0.1000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.1000 \\ 0.0000 & 0.0000 & -0.1200 & 0.0100 \\ 0.0000 & 0.0000 & 0.0100 & -0.1200 \end{bmatrix}, B = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.1200 \\ -0.0100 \end{bmatrix}$
	f_3	$A = \begin{bmatrix} 0.0000 & 0.0000 & 0.1000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.1000 \\ 0.0000 & 0.0000 & -0.1200 & 0.0100 \\ 0.0000 & 0.0000 & 0.0100 & -0.1200 \end{bmatrix}, B = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.0919 \\ -0.0919 \end{bmatrix}$

A.2 The Buck-converter System

Table 7. Guard conditions e and Dynamics f of the learned automata for the buck converter system.

e	$e_1 : 0.0004x_1 - 0.0824x_2 + 1.0 \leq 0$
	$e_2 : -78.8x_1 - 0.0638x_2 + 1.0 \geq 0.0$
	$e_3 : 0.0012x_1 - 0.0842x_2 + 1.0 \geq 0.0$
f	$f_1 : A = \begin{bmatrix} -271.7 & -377.4 \\ 454.5 & -45.45 \end{bmatrix} B = \begin{bmatrix} 9056.6 \\ 0.0000 \end{bmatrix}$
	$f_2 : A = \begin{bmatrix} -195.1 & -378.4 \\ 454.6 & -45.47 \end{bmatrix} B = \begin{bmatrix} 11.29 \\ 0.1235 \end{bmatrix}$
	$f_3 : A = \begin{bmatrix} -0.1511 & 0.6177 \\ 0.0001 & -45.45 \end{bmatrix} B = \begin{bmatrix} -9.392 \\ 0.0057 \end{bmatrix}$

A.3 The Multi-room Heating System

Table 8. Guard conditions e and Dynamics f of the learned automata for the multi-room heating system.

e	$e_2 : -0.4794x_2 + 0.4770x_3 + 1 \leq 0 \ \wedge \ 0.0014x_1 - 0.0659x_3 + 1 \geq 0$	
	$e_3 : 0.4884x_1 + 0.0013x_2 - 0.4901x_3 + 1 \leq 0 \ \wedge \ -0.0627x_1 + 1 \geq 0$	
	$e_4 : -0.3309x_1 + 0.3292x_2 + 0.0016x_3 + 1 \leq 0 \ \wedge \ -0.0641x_2 + 1 \geq 0$	
	$e_5 : -0.0480x_2 + 1 \leq 0$	
	$e_6 : -0.0527x_2 + 1 \geq 0$	
	$e_7 : -0.0456x_3 + 1 \leq 0$	
	$e_8 : -0.0539x_3 + 1 \geq 0$	
	$e_9 : -0.0475x_1 + 1 \leq 0$	
	$e_{10} : -0.0496x_1 + 1 \geq 0$	
f	$f_1 : A = \begin{bmatrix} -0.1001 & 0.0298 & 0.0040 \\ 0.0299 & -0.1001 & 0.0500 \\ 0.0398 & 0.0497 & -0.1400 \end{bmatrix} B = \begin{bmatrix} 0.0301 & 0.0036 \\ 0.0200 & 0.0024 \\ 0.0501 & 1.1060 \end{bmatrix}$	
	$f_2 : A = \begin{bmatrix} -0.0994 & 0.0302 & 0.0392 \\ 0.0209 & -0.1040 & 0.0530 \\ 0.0410 & 0.0504 & -0.1413 \end{bmatrix} B = \begin{bmatrix} 0.0301 & -0.0025 \\ 0.0230 & 0.8405 \\ 0.0502 & -0.0042 \end{bmatrix}$	
	$f_3 : A = \begin{bmatrix} -0.0532 & -0.0776 & 0.0941 \\ 0.0299 & -0.0999 & 0.0500 \\ 0.0397 & 0.0502 & -0.1399 \end{bmatrix} B = \begin{bmatrix} 0.0230 & 0.9161 \\ 0.0200 & 0.0000 \\ 0.0501 & 0.0000 \end{bmatrix}$	
	$f_4 : A = \begin{bmatrix} -0.1003 & 0.0298 & 0.0397 \\ 0.0298 & -0.1001 & 0.0498 \\ 0.0395 & 0.0497 & -0.1406 \end{bmatrix} B = \begin{bmatrix} 0.0303 & 0.0107 \\ 0.0202 & 0.0071 \\ 0.0505 & 0.0179 \end{bmatrix}$	

A.4 Cooperative Vehicles

Table 9. Guard conditions e and Dynamics f of the learned automata for the multi-room heating system.

$e_1 - 0.5128t + 1 \leq 0; t = 0;$	
$e_2 - 0.5128t + 1 \leq 0; t = 0;$	
$f_2 : A =$	$\begin{bmatrix} -0.2474 & 0.5568 & 0.2046 & 0.0583 & -0.2216 & -0.0015 & -0.0227 & -0.1457 & 0.2183 \\ -0.3591 & -1.2851 & -0.8297 & -0.2218 & -0.8613 & -0.0163 & -0.4111 & -0.7054 & 0.7599 \\ 1.3970 & 3.6241 & -2.8374 & -0.2900 & -0.3110 & -0.0552 & -0.2818 & -0.3502 & -0.0988 \\ 0.1469 & -0.0462 & -0.1657 & -0.1852 & 0.8500 & 0.0151 & -0.1603 & -0.1636 & 0.077 \\ -0.1605 & -0.7953 & 0.9879 & -0.2192 & -0.6950 & -1.0086 & -0.3207 & -0.6155 & 0.6032 \\ 0.1304 & -0.3190 & -0.1390 & 0.9189 & 3.1426 & -2.0634 & -0.0656 & 0.3862 & -0.5285 \\ -0.0182 & -0.4770 & -0.0579 & -0.2057 & -0.3947 & -0.0089 & -0.2646 & 0.6382 & 0.3288 \\ -0.1252 & -0.8587 & -0.0594 & -0.2904 & -0.7963 & 1.0033 & -0.3948 & -0.7162 & -0.3455 \\ 0.8712 & 4.1273 & -0.1135 & 0.9544 & 3.6687 & -0.0814 & 1.4677 & 3.4067 & -3.5463 \end{bmatrix}$ $B = \begin{bmatrix} 0.0850 & -0.0487 \\ 1.4159 & 0.3822 \\ 0.3349 & 0.1665 \\ 0.0800 & 0.2523 \\ 0.2593 & 0.2860 \\ 0.8062 & 2.5326 \\ 0.1943 & 0.3559 \\ 0.2998 & 0.3681 \\ -0.1777 & -0.1891 \end{bmatrix}$
$f_3 : A =$	$\begin{bmatrix} -0.1296 & 0.8586 & 0.1423 & 0.0820 & -0.0756 & 0.0058 & 0.0435 & -0.0323 & 0.0255 \\ -0.2269 & -0.4539 & -0.6487 & 0.1232 & -0.1842 & -0.0092 & 0.0259 & -0.1155 & -0.0028 \\ 1.3946 & 4.4506 & -3.1376 & -0.6166 & 0.5344 & -0.0472 & -0.0463 & 0.4700 & -0.2756 \\ 0.1619 & 0.1622 & -0.3123 & -0.1775 & 0.9763 & 0.0094 & -0.1086 & -0.0626 & 0.14073 \\ -0.0717 & -0.2104 & 1.0574 & -0.0124 & -0.1691 & -0.9987 & -0.0480 & -0.1370 & 0.1150 \\ 0.4699 & 3.3837 & -0.0129 & 1.2955 & 3.3234 & -2.2285 & 0.1033 & 0.2572 & -0.7385 \\ 0.0000 & -0.1019 & -0.0034 & -0.0422 & -0.0908 & -0.0166 & -0.0618 & 0.9239 & 0.0792 \\ 0.0113 & -0.0746 & -0.0686 & -0.06537 & -0.1245 & 1.0026 & -0.0680 & -0.12317 & -0.8793 \\ 0.6570 & 3.5501 & 0.0957 & 0.9566 & 3.3547 & -0.0977 & 1.3575 & 3.1740 & -3.2994 \end{bmatrix}$ $B = \begin{bmatrix} 0.0089 & -0.0984 \\ 1.1166 & 0.0102 \\ 0.0762 & -0.0838 \\ -0.0057 & 0.0822 \\ 0.0564 & 0.0566 \\ 0.1114 & -0.3391 \\ 0.0408 & 0.0909 \\ 0.0325 & 0.0744 \\ -0.0176 & -0.0396 \end{bmatrix}$