Hybrid System Identification from Input-Output Traces

Anonymous authors for double-blind review

ABSTRACT

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 black-box 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 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 with state 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. We demonstrate the utility of this framework by evaluating its performance in several case studies as implemented through a prototype software framework.

ACM Reference format:
Anonymous authors for double-blind review. 2018. Hybrid System Identification from Input-Output Traces. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 10 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Black-box 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 [3] now instantiated in software packages such as LearnLib [22]. From an automata theoretic 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 [9–11]. From the automata theoretic guise, one can thus view learning automata from traces as a methodology to infer classes of specifications beyond safety and into temporal behaviors such as liveness.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

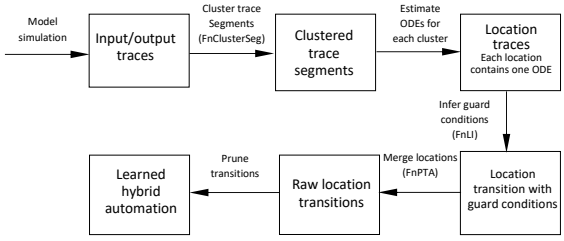

Figure 1: Overview of the framework developed in this paper to learn hybrid automata with linear (affine) ODEs, guards, invariants, and resets from time-series data (traces).

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 [16]. 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 form of graph or state machine. The discrete modes may contain invariant conditions which, once violated, may induce transitions to different modes. The transitions between discrete modes may also be guarded with conditions including exogenous events and predicates over continuous variables.

Hybrid automata provide 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 [2, 17]. 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, such as for runtime monitoring purposes.

Toward this end, in this paper, an automata-based framework for the inference of hybrid systems from execution traces is proposed, with restrictions on the continuous behaviors, guards, invariants, and resets to be described by *linear* equations. This 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 similar locations and transitions in terms of a defined compatibility criterion, and (5) pruning duplicate and other erroneous transitions that may arise from steps 1 and 3. We approximate the continuous dynamics by fitting linear (affine) ODEs to segmented traces, which is a classical

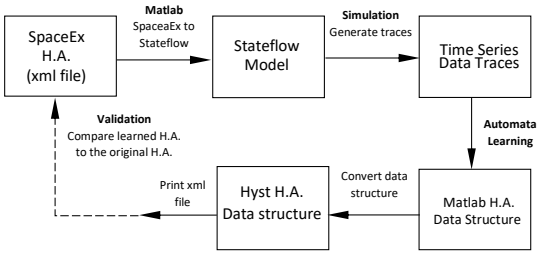

Figure 2: Overview of the validation framework for the method described in this paper and implemented in the software prototype. A given hybrid automaton is specified in the SpaceEx [14] format, automatically translated to a Simulink/Stateflow model using Hyst [5, 6], simulated to generate time series data (traces), the hybrid automata learning methods developed in this paper are then applied for these traces, and a learned hybrid automaton is generated in the Hyst format, and simulated again generate traces to validate the learned models behaviors against the original models behaviors.

problem in *system identification* [18]. Instead of estimating the optimal solution for each segment, we propose a method to calculate their solution spaces within a pre-specified error bound and cluster their dynamics by inspecting the intersection of these solution space. 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 the changepoints into a low-dimensional *flat*, that is, an n-dimensional line or plane. For the location merging, a method based on the prefix tree acceptor (PTA) is applied to merge similar location without introducing non-determinism into the model, and erroneous transitions will be pruned before generating the final hybrid automaton model 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 [6] and its integration with Simulink/Stateflow [5], and is evaluated and validated against several standard hybrid systems benchmarks, such as the multi-room heating system [12]. the navigation system [12], and the bouncing ball. These examples were chosen in part so that the validation approach illustrated in Figure 2 could be pursued, where both syntactic and semantic similarities and differences between the original model generating the traces could be compared to the learned automata, but the framework is applicable to general time-series traces as inputs, that possibly could be generated from black-box systems.

2 RELATED WORK

Significant related work has been developed in the context of automata learning for *purely discrete* systems, such as finite state automata, but less research has investigated learning models from time-series data that incorporate continuous behaviors, such as timed automata or hybrid automata. Among these methods however, some have been developed to estimate the continuous and discrete dynamics of hybrid systems from observed behaviors. Summerville et al. proposed a framework for hybrid automaton inference [24], where a cost function with a penalty criterion based on one set of potential linear model templates is applied to segment the traces and select an optimal model that minimizes error between the executions of the learned model and the original traces. For guard conditions, this work applied Normalized Pointwise Mutual Information (NPMI) to select predicates from a predefined set for each mode transition.

Instead of a predefined set of potential model templates, Niggemann et al. model each segment using linear regression or neural networks [21]. The similarity of two states is tested by checking the probability of staying in the state or of a specific transition occurring. Then, states are merged in a bottom-up fashion with higher efficiency compared merging in a top-down order. Guard conditions are estimated by a combination of exogenous events, timing constraints, and transition probabilities.

In contrast to Niggemann's work, Medhat et al. cluster the observed traces into input/output events according to predefined features [19], then linear regression is used to estimate the dynamics of the clustered trace segments. They derive an automaton based on a Mealy inference algorithm within LearnLib. However, the internal guard conditions that trigger state transitions are not taken into consideration.

Grosu et al. propose a methodology to estimate cycle-linear hybrid automata for excitable cells from virtual measurements [15]. The traces of electrical signals with respect to time are first 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, in their case a transition voltage for mode switches, is estimated from the transitions' post states. Finally, all of the derived hybrid automata are merged.

All of these existing methods deal with guard conditions that only contains exogenous event and lack an estimation of internal event such as guard conditions being defined by predicates such as linear inequalities being satisfied, which may reduce the general applicability and scalability of these existing frameworks. Additionally, many of the existing methods for merging discrete locations also suffer from generalizeability and scalability issues.

3 HYBRID AUTOMATA

Hybrid automata are a common formal modelling framework for hybrid systems, which combine a finite state machine with a finite set of real-valued continuous state variables whose continuous time evolution is described by differential equations or inclusions. Our work mainly focuses on the deterministic and synchronous model where there are no time-triggered transitions, and all constraints over real variables are specified using linear (affine) equations or inequalities.

Provided a set of time series traces, possibly generated from the executions of a hybrid system, a formal inference model, G_h, for this hybrid system is defined as a hybrid automaton. According to this definition, the system dynamics in each mode can be characterized by a three-tuple $\mathcal{F} = \langle A, B, u_c \rangle$, where A and B correspond to the standard state matrix and input vector of a linear ODE and

u_c is an input, which together will be used for the further mode identification, state merging, etc.

Definition 3.1 (Hybrid System Model). Let the hybrid automaton be denoted by G_h, which is a six-tuple $G_h = \langle Q, X, f, E, \Phi, U \rangle$:

- Q is a finite set of control *modes* (or locations), and $X \subseteq \mathbb{R}^n$ is the continuous state space, an element of which is typically denoted as $x \in X$.
- $U \subseteq \mathbb{R}^m$ denotes a continuous space of inputs, and $u \in U$ is a input consisting of exogenous continuous input, $u_d \in \mathbb{R}^{m_d}$, and a local constant, $u_c \in \mathbb{R}^{m_c}$, in each mode, used to characterize differences between modes.
- f is a vector field that describes the dynamics of X with respect to real-time, $f : Q \times X \times U \rightarrow X$. For a mode $q \in Q$, we define f as follows, where $A_q \in X \times X$ and $B_q \in X$ are time-invariant within mode q,
$$\dot{x} = f(q, x, u) = A_q x + Bu.$$
- E denotes events that trigger state transitions, where $e \in E$ is an exogenous event and multiple linear inequalities (predicates) involving continuous variables may appear.
- Φ denotes the discrete transitions: $Q \times E \rightarrow Q$. Here, $\phi : \langle q, e, q' \rangle \in \Phi$ denotes a mode transition from source mode q to destination mode q' triggered by e.

An illustration of the structure of the inference model is shown in Figure 3. The input I consists of the continuous input u_d which can be control signal or exogenous disturbance, and discrete events *event* triggering state transitions, and the model if learned properly can reproduce the original traces.

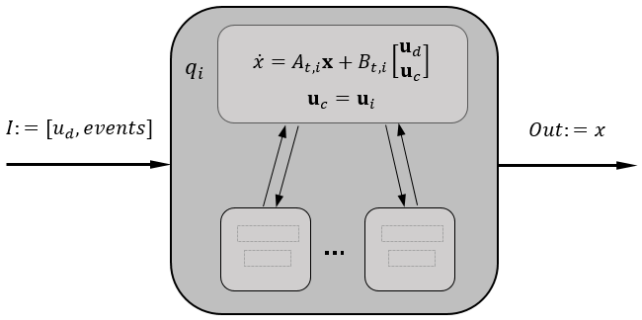

Figure 3: Illustrative model of an inferred automaton.

4 IDENTIFYING HYBRID AUTOMATON MODES FROM TRACES

In this section, we present a method to estimate and cluster ordinary differential equations (ODEs) from traces. We assume two consecutive points along a trace in which an abrupt change occurs (i.e., a discontinuity) may have different dynamics. Therefore, the abrupt change and discontinuity can reflect a mode switch. A parameter space $\mathcal{A}$ of the ODE for each segment of the trace is estimated with a pre-specified error tolerance. Subsequently, trace segments are clustered by checking if their parameter spaces have an intersection.

4.1 Input-output Traces and Examples

The dynamics of hybrid systems are reflected in the behaviors of execution traces. One such trace is one set of a finite sequence of input signals and their corresponding output values (or state variable values). First, the signal format is defined. Let $\mathcal{X} = [\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_p]$ describe that the output trace is divided into p segments and the continuous states be represented by $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ where n is the number of the continuous states. Similarly, let $\mathcal{U} = [\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_p]$ represent the segmented input and the input be $\mathbf{u} = [u_1, u_2, \ldots, u_m]$, where m is the number of the input.

This definition of traces is demonstrated by one example of temperature traces collected from the multi-room heating system benchmark, as show in Figure 4.

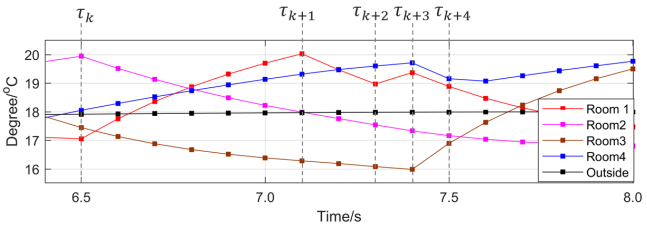

Figure 4: One input/output trace from a multi-room heating system. This set of data includes the temperature measurement of four adjacent rooms and one exogenous temperature from outside environment, and here τ denotes a discontinuity, henceforth referred to as a *changepoint*.

4.2 Changepoint Detection

The changepoint describes the abrupt variations in the output traces, which captures state transitions. By segmenting the output traces into smaller data sets, the dynamics of a hybrid system in each mode can be extracted and the ODE can be derived. Therefore, changepoint detection plays an important role in the inference of hybrid automata. A significant number of well-performing algorithms for changepoint detection have been developed for a data set. In addition, the development of a robust changepoint detecting method is out of the scope of this work. Therefore, we assume that the input-output traces have been well segmented before its application. Let $\tau = [\tau_1, \tau_2, \ldots, \tau_p]$ be the obtained timestamp changepoints and τ_s be the sampling interval.

4.3 Identification of Parameter Space

The discrete time representation of f is

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}. \tag{1}$$

Let the matrix $B = [B_d, B_c]$ where B_d and B_c respectively correspond to the input u_d and u_c. Since both the matrix B_c and input u_c are unknown, it is reasonable to consider their multiplication as one variable $B_u = B_c u_c$. Thus, equation 2 for the trace segment $\mathcal{X}_k$ can be derived from equation 1.

$$O_k^{'} - \mathcal{A}_k O_k = 0 \tag{2}$$

where $O_k^{'} = \mathcal{X}_{k,[\tau_{(k-1)}+\tau_s, \tau_k]}$, $O_k = [\mathcal{X}_k^{\mathsf{T}}, \mathcal{U}_d^{\mathsf{T}}, \mathbf{1}]^{\mathsf{T}}_{[\tau_{(k-1)}, \tau_k - \tau_s]}$ and $\mathcal{A}_k = [A, B_d, B_u]$.

Note that the only unknown variable vector is $\mathcal{A}$, whose optimal solution can be easily approximated through the method of least square. However, the optimal parameters of trace segments that are generated from the mode may have an unpredictable difference, which may lead to the failure of clustering. Therefore, instead of calculating and clustering the optimal solution for each trace segment, a method for estimating a solution space within an error bound and clustering them through their intersection is proposed here. Accordingly, equation 2 is modified to that

$$\forall i \in [1, 2, \ldots, n], \quad \frac{1}{l}\|\{O_k^{'} - \mathcal{A}O_k\}_i\|_2 \leq \sigma$$

by adding an error tolerance, σ, where l denotes the trace length and $\{*\}_i$ denotes the trace of ith variable. Then, the solution space of $\mathcal{A}$ for data segment X_k can be denoted by

$$\mathcal{S}_k = \{\mathcal{A} | \forall i \in [1, 2, \ldots, n], \ \frac{1}{l}\|\{O_k^{'} - \mathcal{A}O_k\}_i\|_2 \leq \sigma\}.$$

4.4 Intersection of Solution Space

In this section, methods for linear matrix inequalities (LMIs) are applied to determine the existence of the intersection between two different solution spaces. The existence of an intersection implies that the corresponding data segment could be clustered into one mode. According to Theorem 4.1, which has been proved in the work [27], the parameter space $\mathcal{S}_k$ is equivalent to the feasible solutions of a non-strict LMI that $\forall i \in [1, 2, \ldots, n]$,

$$\begin{bmatrix} I & (O_k^{'} - \mathcal{A}O_k)^\intercal C_i^\intercal \\ C_i(O_k^{'} - \mathcal{A}O_k) & (l\sigma)^2 \end{bmatrix} \geq 0. \tag{3}$$

where C_i is to select the trace of ith variable.

Given a feasibility, $\mathcal{A}_f$, such that $F(\mathcal{A}_f) \succeq 0$ or determine that the LMI is infeasible. For the multiple LMIs $F(\mathcal{A})_1 \succeq 0$, $F(\mathcal{A})_2 \succeq 0$, $\ldots$ $F(\mathcal{A})_p \succeq 0$, they can be described by a single LMI:

$$\mathbf{diag}\{F(\mathcal{A})_1, F(\mathcal{A})_2, \ldots, F(\mathcal{A})_p\} \succeq 0.$$

Here we apply the polynomial-time projective method proposed by Nemirovskii [20], which is one of the most efficient algorithms among interior-point methods for solving LMI problems.

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

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

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

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

The method of identifying modes is illustrated in algorithm 1, which recursively searches for the trace segments belonging to the same mode by determining if their solution spaces intersect. In practice, the solution space intersection between smaller trace segments in different categories, may exist and misclassification will be propagated through the intersection calculation of following segments, and may result in a failure of clustering the segments. So, we assume that the longer trace segment carries more information and sort function's input into one set of trace segments in decreasing order according to their data length. The function's output is the clustered index of trace segments. Given one trace segment, the function `FnSoluspace` returns its solution space expressed in LMIs. The item $\mathcal{I}$ represents the solution space in LMI. The function `FnCombine` combines two solution spaces. The function `FnInspace` is used to calculate the feasibility of multiple LMIs through the projective method. The negative value indicates that the intersection exists between the given multiple solution spaces. In this case, the trace segment's index will be added to the current cluster. Otherwise, it will be collected for further clustering. After clustering, all the data in the same category will be applied to calculate the parameter matrix $\mathcal{A}$. The achieved ODEs will be labelled and then the segmented trace is converted into a ODE-label trace. By combining two adjacent labels and the changepoint between them in the trace, we can create an ODE transition where the label only indicates the dynamics in corresponding data segment and does not indicate a unique mode. In this transition, we name the first label the source ODE, and the second label the destination ODE.

Algorithm 1: Identification of modes

```
O ← p sets of sorted trace segments;
Function FnRecursive(O)
    O_c, I_n ← empty ;
    if O ≠ empty then
        I_1 ← FnSoluspace(O_1);
        for j = 2 to p do
            I_j ← FnSoluspace(O_j);
            I_temp ← FnCombine (I_1, I_j);
            feasibility ← FnInspace (I_temp) ;
            if feasibility < 0 then
                j add to I_n ;
            else
                O_j add to O_c;
            end
        end
        FnRecursive (O_c) add to I_n;
        return I_n;
    else
        return empty;
    end
end
```

5 INFERRING HYBRID AUTOMATA FROM HYBRID SYSTEMS TRACES

After creating the discrete modes, methods of estimating guard conditions and state transitions are proposed in this section. There exists some work that aims to reconstruct a deterministic finite machine or Mealy machine having similar behaviors to the target system [7, 22, 23], where systems are treated as black boxes and state transitions are only triggered by exogenous events. In our method, we also consider the internal guard conditions that are expressed in linear inequalities (LIs). By estimating the LI, we can map each changepoint in ODE traces to a guard condition in LI or an exogenous event, and thus construct ODE transitions for further merging of states.

5.1 Inference of Guard Conditions

As described in definition 3.1, there are two types of guard conditions: exogenous *event* and LIs in $\mathbb{R}^n$, where n indicates the number of continuous variables. Since the model is assumed synchronous and there is no delay between the input and output, the *event*'s immediate impact on the system will be directly reflected in a state transition and its corresponding changepoint in output traces, from which the association between the *event* and its state transition can be achieved. In this section, we mainly focus on the estimation of the LIs.

Given changepoints, the LI estimation can be converted into affine-subspace clustering, which aims to cluster data into multiple low-dimensional flat. To approximate the flat, we apply Random Sample Consensus (RANSAC), a statistical method proposed in [13]. It is a learning technique for estimating parameters of a mathematical model by iteratively and randomly sampling a set of observed data. The observed data are assumed to have inliers, points that can be approximated by fitting to a flat, and outliers, points that cannot be fitted. The model that contains the most inliers is selected as the optimal one. In our case, the changepoints are all close to the boundary of LIs and there will not be many outliers in the data points. RANSAC deals with such data sets with a very competitive performance compared with other methods [4].

The original RANSAC only estimates one flat for one particular data group. However, a guard condition may include more than one LI, which means there may exist multiple flats to estimate. Inspired by the work [25, 26], we choose to apply RANSAC, sequentially, to mine a new subspace from the modified data set, where the points belonging to previously found models are removed. However, a large number of flat in the data set will possibly result in a poor estimation, because it has a high chance that points belonging to different flats are chosen and misclassified. To reduce its number in one data set, we split changepoints by clustering the ones having the same source ODE as well as the same destination ODE into one data set.

For each clustered changepoints, we execute the algorithm that is shown in algorithm 2. *inData* and *inNum* denote the inliers and its number, respectively. During each iteration, the function `FnRandomSample` randomly selects a model candidate. The function `FnValidP` finds all the points that are compatible with that model. Once a new model is determined, the corresponding points will be removed before the next iteration. The process will be terminated after no more models can be found.

Similar to the original RANSAC, the modified one has three parameters to specify: (1) error tolerance, ε, 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, ε, 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 flats, there should be an error tolerance, ε, such that most of the points become inliers. Decreasing, iteratively, from a large value candidate to approach such a ε helps achieve a robust estimation. For the threshold number,

Algorithm 2: Identification of Multi-flats

```
while true do
    bestModel, bestNum, bestData ← empty ;
    Remove(bestData);
    for n = 1 to iter do
        model ← FnRandomSample(data);
        inNum, inData ← FnValidP(model);
        if inNum > num & inNum > bestNum then
            bestNum, bestData ← inNum, inData;
            bestModel ← model;
        end
    end
    if bestModel ≠ empty then
        Output (bestModel);
    else
        break
    end
end
```

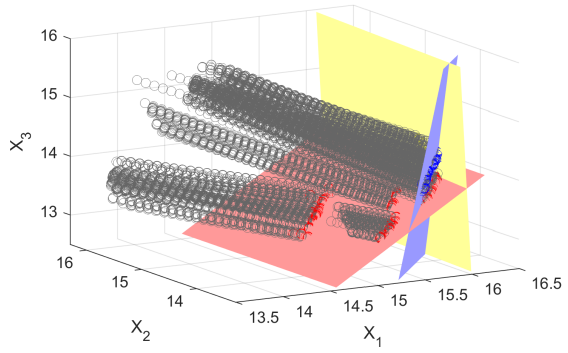

Figure 5: Linear inequality estimation and data trajectory.

suppose that the probability of one point being compatible with all the 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 [13], 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.

Figure 5 demonstrates one sample of linear inequality estimation for a state transition that is involved in the temperature dynamics of a multi-room heating system. The temperature data are collected from several output traces. Each circle trajectories in grey describe the temperature dynamics when staying in one discrete mode. The colored trajectories describe the changepoints, from which multiple colored flats are estimated using the modified RANSAC. The trajectories' positions, with respect to their flats, are used to determine the inequality sign. The logical connective between multiple linear inequalities that are estimated from one changepoint set will also be taken into consideration. The conjunction is determined if all

the linear inequalities are satisfied by the set, otherwise, each linear inequality will be treat as an individual guard condition. For the changepoints that are treated as outliers during the estimation, we use a label 0, instead of a LI, to indicate the guard condition of the corresponding ODE transitions.

5.2 Merging of Locations

In this section, we introduce the concept of the prefix tree acceptor (PTA) here to help merge the ODE traces from the previous section and construct the final hybrid model. As shown in Figure 6, each subtree in a PTA represents one processed output trace. Each state is characterized by their own ODE, f, and the guard condition in each state transition is denoted by e. Unlike the work [8, 21] shows that the similarity of two states is associated with the probability of staying in or transitioning out of a state. In our method, since the dynmaics of each state is already known, the compatibility of two states can be directly evaluated by the derived ODEs. To avoid creating a non-deterministic system during the process of merging states, states are determined compatible in the following situations:

- The source ODE, destination ODE and guard conditions of two transitions are the same, and meanwhile, in the subsequent transition where the destination ODE are the next source ODE, there doesn't exist different transitions that are triggered by the same guard condition. Then the states involved in these two transition are compatible. This situation is illustrated in Figure 6.
- The first states in each trace are compatible because the initial mode is fixed when the original system generating traces, and the first segments in each trace represent the same mode.

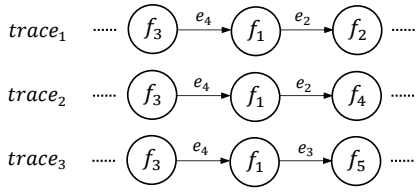

Figure 6: Illustration of state merging. According to the compatibility criterion, the f_3 state and the f_1 state in $trace_1$ are respectively compatible with the f_3 state and the f_1 state in $trace_3$. However, they are not compatible with the states in $trace_2$ because in the subsequent transition, the guard condition e_2 triggers different transitions from the f_1 state.

In the algorithm of merging states, we first construct a 6-tuple $\langle label_1, e, label_2, id_1, id_2, times\rangle$ for each ODE transition in all the traces. The $label_1$ and $label_2$ respectively denotes the labels of the source and destination ODE. Item e denotes the guard condition. The ODEs in all the traces will be assigned an unique index id which indicates one discrete mode in the hybrid automaton. The id_1 and id_2 denote the index of the source and destination ODE. 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 `FnCompatiblity`. If they are compatible, we apply the function `FnModifyTuple` to modify all the ODE transitions that contains the index id_1 or id_2 of the $jtuple$ by changing that index to the corresponding index in $ituple$. Thus the connectivity between transitions can be maintained. Afterwards, the $jtuple$ is emptied and the $times$ in $ituple$ is increased by one.

Algorithm 3: Merging of States

```
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 FnCompatiblity(ituple, jtuple) then
                FnModifyTuple(ituple, jtuple);
                tuple(j) ← empty ;
                ituple.times + +;
            end
        end
    end
end
return tuples;
```

6 CASE STUDIES AND VALIDATING THE HYBRID AUTOMATA LEARNING FRAMEWORK

In this section, we apply three case studies to evaluate the proposed hybrid automata learning framework on different systems, following the validation overview from Figure 2. The chosen systems all have linear ODEs, guards, invariants, and resets. The first two systems, the navigation system and multi-room heating system, are standard benchmarks [12]. 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 [6] and its integration with Simulink/Stateflow [5], which is available here: https://www.dropbox.com/sh/wlqwdm1i38zm0xz/AAARv1g8b5jqj9VUtRt9msBKa?dl=0.

6.1 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 we aim to learn is shown in Figure 7. Given the desired velocity $\mathbf{v}_d$, the dynamics of the actual velocity $\mathbf{v}$ is described by the differential equation $\dot{\mathbf{v}} = A(\mathbf{v} - \mathbf{v}_d)$. To guarantee that the actual velocity $\mathbf{v}$ will converge to the desired velocity $\mathbf{v}_d$, the eigenvalues

of A should have negative real part. In our case, we set

$$A = \begin{bmatrix} -1.2 & 0.1 \\ 0.1 & -1.2 \end{bmatrix}.$$

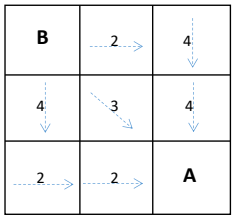

Figure 7: Navigation system in a 3×3 grid. The label i refers to the parameter to calculated the desired velocity. The object needs to reach the grid labelled A and meanwhile avoid the grid labelled B.

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 $[x, y]$. From the trajectories, we can also derive the velocities $[v_x, v_y]$. Thus, $[x, y, v_x, v_y]$ are treated as four continuous variables, and a hybrid automaton will be estimated through our method to approximate their dynamics. We estimated one hybrid automaton for the traces starting from the lower left grid. For the learning, we set $\sigma = 10^{-6}$ for clustering the trace segments, $\varepsilon = 0.01$, $\eta = 1000$ and $\gamma = 5$ for estimating the LIs. Here, 68 traces are collected with a sampling time $t_s = 0.1s$, and the estimated state transition are listed in Table 1.The transition in red has the same source and destination ODEs and the label of its guard condition is 0. Two identical ODEs are learned due to mis-clustering two sequential trace segments together, and the label 0 occurs because their changepoints fall in outliers during the estimation of LIs, and no LIs are estimated for them. Therefore, the transition in red dashed line is deleted from the final hybrid automaton based on the guard condition being 0. The information for labels e and f are listed in Table 2. Based on this information, a hybrid automaton is constructed as shown in Figure 8.

6.2 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

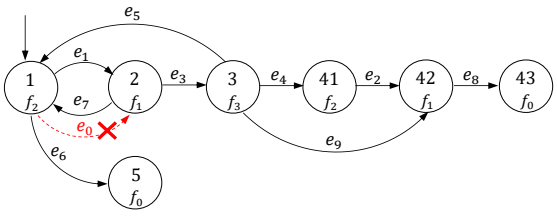

Figure 8: Estimated hybrid automaton for the first grid.

Table 1: Estimated state transitions expressed in a list of 6-tuples

$label_1$	$guard$	$label_2$	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	41	29
2	2	1	41	42	29
1	8	0	42	43	42
3	9	1	3	42	13
2	0	1	1	2	2

Table 2: The information of the label e and f for the navigation benchmark.

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	$A = \begin{bmatrix} 0.0000 & 0.0000 & 0.1000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.1000 \\ 0.0000 & 0.0000 & -0.1200 & 0.0100 \\ 0.0000 & 0.0000 & 0.0100 & -0.1200 \end{bmatrix}, B_d = 0, B_c = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.0100 \\ -0.1200 \end{bmatrix}$
	f_2	$A = \begin{bmatrix} 0.0000 & 0.0000 & 0.1000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.1000 \\ 0.0000 & 0.0000 & -0.1200 & 0.0100 \\ 0.0000 & 0.0000 & 0.0100 & -0.1200 \end{bmatrix}, B_d = 0, B_c = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.1200 \\ -0.0100 \end{bmatrix}$
	f_3	$A = \begin{bmatrix} 0.0000 & 0.0000 & 0.1000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.1000 \\ 0.0000 & 0.0000 & -0.1200 & 0.0100 \\ 0.0000 & 0.0000 & 0.0100 & -0.1200 \end{bmatrix}, B_d = 0, B_c = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.0919 \\ -0.0919 \end{bmatrix}$

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 the other rooms, which is described by

$$\dot{x}_i = c_i h_i + b_i(u - x_i) + \sum_{j \neq i} a_{i,j}(x_j - x_i), \tag{4}$$

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) there is not a heater in room i, (2) there is a heater in room j, (3) $x_i \leq get_i$, and (4) $x_j - x_i \geq dif_i$.

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 thus the system becomes nondeterministic, we restrict that there is only one destination room for each source room. Then, the system controller can be modelled

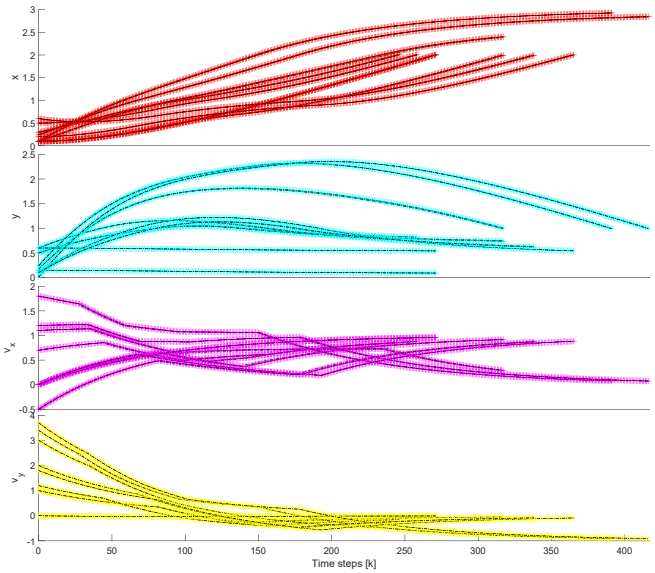

Figure 9: Traces from the learned hybrid automaton for the navigation benchmark and their errors from the training trace data at discrete time steps, a subset comparing 10 of the 68 traces learned and validated against is shown.

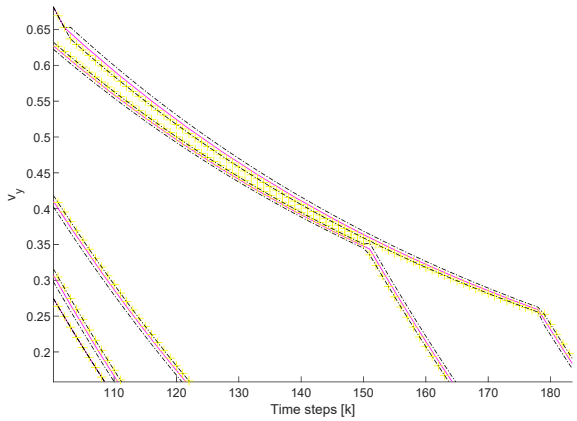

Figure 10: A zoomed version of Figure 9 for v_y illustrating the tight error bounds between the training trace data and executions of the learned hybrid automaton. For clarity, not all 68 traces are shown, but the overall mean-squared error (MSE) between each training trace and the corresponding learned hybrid automaton's trace ranges from $4.33e-7$ to 0.32, with a mean of 0.11 and median of 0.11, which as the tightness of the traces indicates, is fairly negligible error.

as shown in Figure 11. Its input and output are, respectively, the feedback from temperature dynamics $[x_1, x_2, x_3]$ and the power status of the heater $[h_1, h_2, h_3]$. Combining the controller and the plant in equation 4, we achieve the final system model which has **u** and $[x_1, x_2, x_3]$ as the input and output, respectively.

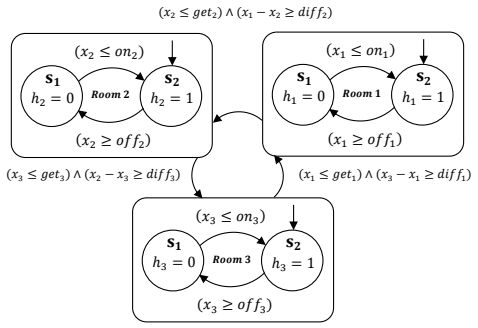

Figure 11: Controller for the heating system.

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

The input/output traces are collected by running simulations in Matlab with a sampling interval of 0.1 seconds, through which an approximated hybrid automaton is generated with our framework. The raw state transitions in 6-tuples generated from the framework is shown in Table 3. The transition in the first row has the same source ODE and destination ODE, which is also treated as a bad state transition. Such a transition occurs because the two sequential trace segments are clustered together. The reason why the transitions have a guard condition labelled âĂIJ0âĂİ is that their corresponding changepoints are regarded as outliers in the LI estimation. These bad transitions will be pruned before generating the final hybrid automaton. The final hybrid automata is shown in Figure 12, which has 6 discrete mode, 4 distinct ODEs, and 9 mode switches in total. The initial state is the mode 2. The parameters of the guard conditions (event labels) and ODEs are as following:

To verify the hybrid automaton, we compare its output traces with the traces from the original system. As shown in Figure 13, there exist good matches between their dynamics. But there are also some delays between the corresponding changepoints, and these delay gets larger over time, which will ultimately lead to a mismatch. One reason is that the difference between the estimated linear

Table 3: Estimated hybrid automaton for the heating system.

$label_1$	$guard$	$label_2$	id_1	id_2	$times$
2	1	2	1	1	2
2	2	1	2	4	107
1	3	3	4	5	103
3	4	2	5	2	103
1	0	5	4	227	1
5	0	2	227	2	1
2	0	3	1	234	4
3	0	2	234	2	3
2	5	4	2	236	25
4	6	2	236	2	25
1	7	4	4	262	99
4	8	1	262	4	96
3	9	4	5	267	57
4	10	3	267	5	57

inequalities and the actual ones will be unavoidably propagated and accumulated throughout the system's performance, having an increasing impact on the state transitions. Another reason is that a comprehensive automaton structure may not be fully recovered from input/output traces.

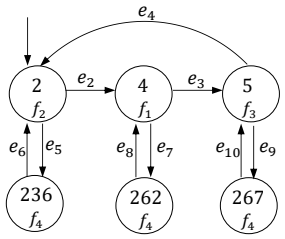

Figure 12: Estimated hybrid automaton of the heating system.

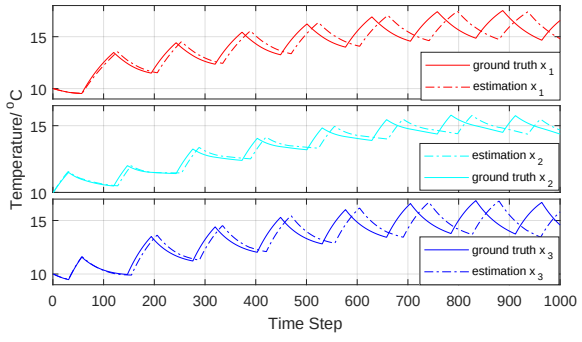

Figure 13: Comparison between the ground truth and the estimated traces.

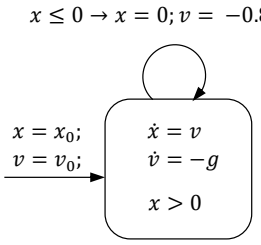

Figure 14: The hybrid automaton model for the bouncing ball system. It has ordinary differential equations for the continuous variables x and v, and it has one self-transition with variable resets. The only invariant condition is $x \geq 0$.

Table 4: Estimated state transitions expressed in a list of 6-tuples.

$label_1$	$guard$	$label_2$	id_1	id_2	$times$
1	1	1	1	1	20

6.3 Bouncing Ball

The bouncing ball model is a classic instance of a hybrid system. The continuous dynamics of the bouncing ball is described by:

$$\frac{dv}{dt} = -g; \quad \frac{dx}{dt} = v$$

where $x(t)$ and $v(t)$ are the position and velocity of the bouncing ball, and g is the acceleration caused by the gravity. The hybrid automaton used to describe this model is shown in Figure 14.

To learn this hybrid system, 10 output traces are collected in a sampling time $t_s = 0.1s$. For the learning, we set $\sigma = 10^{-5}$ for clustering the trace segments, $\varepsilon = 0.001$, $\eta = 1000$ and $\gamma = 5$ for estimating the LIs in guard conditions. To handle the variable resets in during a state transition, we will to check the existence of abrupt change in the variable traces by setting a threshold to the difference between sequential values in the variable trace. The estimated state transitions are listed in Table 4. We can notice that the source state and destination state are the same in the transitions, which indicates there is only one self-transition in this system. By combining the obtained guard condition, we construct the learned automation. The information of the label e and f are as following:

$$e_1 : 7.2511 \times 10^{15} x_1 - 0.0028 * x_2 \leq 0$$

$$f_1 : A = \begin{bmatrix} 0.0000 & 1.0000 \\ 0.0000 & 0.0000 \end{bmatrix}, \quad B_c = \begin{bmatrix} 0.0000 \\ -9.810 \end{bmatrix}$$

, and the reset function of the continuous variables is

$$x = \begin{bmatrix} 0.000 & 0.000 \\ 0.000 & -0.7999 \end{bmatrix}$$

Compared to the parameters in the original system, it can be concluded that the e and f are well recovered, and our framework's ability of handling variable reset in state transitions is demonstrated in this case study.

7 CONCLUSION

This paper presents a framework to mine and learn hybrid automata with linear (affine) constraints and ODEs from input and output traces. For a system that generates outputs with more than one dimension, the framework will mine one hybrid automaton for each dimension. 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, and subsequently used in the state-merging phase. 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 system by exploring methods to estimate and cluster 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. Incorporating more sophisticated measures to validate the framework, such as conformance degree [1], will provide superior quantitative measures on the accuracy of the learned automaton versus the traces used to learn it, relative to our usage of mean squared error (MSE).

REFERENCES

[1] Houssam Abbas, Bardh Hoxha, Georgios Fainekos, Jyotirmoy V Deshmukh, James Kapinski, and Koichi Ueda. 2014. Conformance testing as falsification for cyber-physical systems. *arXiv preprint arXiv:1401.5200* (2014).

[2] 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.

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

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

[5] 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).

[6] 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.

[7] Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker, Daniel Neider, and David R Piegdon. 2010. libalf: The automata learning framework. In *International Conference on Computer Aided Verification*. Springer, 360–364.

[8] 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.

[9] 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

[10] 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

[11] 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.

[12] Ansgar Fehnker and Franjo Ivančić. 2004. Benchmarks for hybrid systems verification. In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 326–341.

[13] 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.

[14] 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.

[15] 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.

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

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

[18] 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

[19] 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.

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

[21] 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.

[22] 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.

[23] Muzammil Shahbaz and Roland Groz. 2009. Inferring mealy machines. In *International Symposium on Formal Methods*. Springer, 207–222.

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

[25] 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.

[26] 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.

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