Hybrid Model Learning for System Health Monitoring

Amaury Vignolles, Elodie Chanthery, Pauline Ribot

Hybrid Model Learning for System Health Monitoring

Amaury Vignolles[1,2], Elodie Chanthery[1,2] and Pauline Ribot[1,3]

***Abstract*— Obtaining a relevant model for a complex system is striven for in the Health Management community, as it allows to precisely monitor the health state of the system. This article introduces a method to obtain a hybrid model under the Heterogeneous Petri Net formalism for a system, using only data. The method is comprised of two steps, the learning of the Discrete Event System (DES) structure of the system using a clustering algorithm, then the learning of the continuous dynamics contained in the system and its various functioning modes using two regression algorithms. The method is applied on an academic example.**

I. Introduction

Health monitoring consists in estimating and predicting the health state of a system. The concept of health state is linked to either the degradation of a system or the occurrence of a fault. A fault is a non-acceptable deviation from a characteristic property or a parameter of the system. It is unobservable by nature: its occurrence is undetected by the sensors. System health monitoring is performed by integrating diagnosis and prognosis capabilities. Diagnosis aims at detecting a fault occurrence and precisely locating it in the system. Prognosis aims at predicting fault occurrences in order to determine the remaining useful life of the system before failure. Model-based health monitoring obviously requires a precise model of the system dynamics describing its behavior and its degradation. However, having a precise model is yet an utopia for most systems, and is what the health management society is striving for. Nowadays, most models are either imperfect, incomplete or nonexistent and solely data is available. When looking at the latter, a system for which a sufficient amount of data is available or obtainable is considered. This amount of data can lead to a model to perform health monitoring or the improvement of an existing one. The studied systems are complex systems and deal both with continuous and discrete data. Their behavior can be represented by a multimode representation by using for example hybrid automaton [Henzinger, 2000]. For health monitoring purpose, a functioning mode of the system is the combination of one discrete state with continuous and degradation dynamics. This multimode representation can then be transformed into an Heterogeneous Petri Nets (HtPN) model. The HtPN formalism is an extension of the Petri Net formalism where the places represent the different functioning modes with associated continuous and degradation dynamics. This formalism was chosen to perform hybrid system health monitoring under uncertainty because of its capability to represent the evolution of different hypotheses on the hybrid system health state thanks to parallelism. However, the learning process could be applied to other formalisms for hybrid systems like hybrid automata.

This paper proposes a methodology to learn a hybrid model for system health monitoring using the formalism of HtPN. The first step of the methodology consists in obtaining the Petri Net structure, represented by places and transitions, using a clustering method. In an HtPN, each functioning mode is associated to continuous and degradation dynamics. Hence, for each of the modes, regression algorithms such as Support Vector Regression (SVR) and Random Forest Regression (RFR) are applied to learn continuous dynamics of the system.

The paper is organized as follows. Section 2 gives a state of the art of existing methods used to learn models of different types (hybrid, discrete or continuous). Section 3 presents the HtPN formalism used for health monitoring purposes. Section 4 presents the proposed learning methodology to obtain a hybrid model under the HtPN formalism. Section 5 illustrates the application of the methodology and gives results for an academic example, the water tanks. Finally Section 6 presents some conclusion and perspectives.

II. Related work

A. Learning a hybrid model

To learn a hybrid model, different methods can be used. In [Lauer and Bloch, 2008], switched and non linear hybrid systems are learned using the Support Vector Machine (SVM) and Support Vector Regression (SVR) frameworks. Both of these techniques are based upon kernels. The SVR algorithm tries to find a function $w_i x_i$ such that the highest number of data points are contained in the interval $[w_i x_i - \epsilon, w_i x_i + \epsilon]$, with w_i the coefficient of the model, x_i the features of the data, and $\epsilon \in \mathbf{R}$ the desired error. A new regularized technique for the identification of piecewise affine systems is proposed in [Pillonetto, 2016]. This technique, named Hybrid Stable Spline algorithm (HSS) uses the stable spline kernel to model the impulse responses of submodels as zero-mean Gaussian processes. [Feng et al., 2010] focuses on identifying discrete time affine hybrid systems with measurement noise. The proposed approach is based on recasting the problem into a polynomial optimization form and exploiting its inherent sparse structure to obtain computationally tractable problems. However, the learned systems in these articles have one major drawback: they cannot include parallelism. As the final objective of our work

*This work was not supported by any organization

[1]All authors are with CNRS, LAAS 7 avenue du colonel Roche, F-31400 Toulouse, France amaury.vignolles@laas.fr elodie.chanthery@laas.fr pauline.ribot@laas.fr

[2]Amaury Vignolles and Elodie Chanthery are with INSA Toulouse, F-31400 Toulouse, France

[3]Pauline Ribot is with UPS, F-31400 Toulouse, France

is to monitor the system health state under uncertainty, it requires to follow different hypotheses: the parallelism is thus a necessary feature in the learned model.

Another possibility to learn hybrid models is obviously to merge techniques used to learn discrete event models and techniques used to learn continuous dynamics.

B. Learning a discrete event model

To learn the structure of a DES, event based methods are mostly used. An example to learn a Petri Net (PN) can be found in [Dotoli et al., 2008]. Using the observations of the events and the available output vectors, a PN modeling the DES is obtained through an integer linear programming problem or an identification algorithm depending on whether the sets of places and transitions are known beforehand. Another example can be found in [Moreira and Lesage, 2019] where a Deterministic Automaton with Outputs and Conditional Transitions (DAOCT) is computed using observed fault-free paths. However, these learning methods fits only for DES, where all data are discrete events, but not for hybrid systems where continuous conditions may induce a health state change. Thus, we had to look at more complex ways to learn a DES.

With a clustering method, both continuous dynamics variations and discrete events can impact the creation of the DES structure. However, not all methods are usable in our case. For instance, clustering methods requiring a known number of outputted clusters, such as K-means, cannot be used. As a matter of fact, even if the size of clusters has to be decided, a fixed number of cluster would be limiting, where a size could lead to improvement of the knowledge on the system. To meet this need, the DyClee algorithm has been proposed for tracking evolving environments with dynamic clustering. DyClee is presented in detail in [Roa et al., 2019]. It is a two stage clustering algorithm, one based on distance and one on density. Each sample is assigned to a micro cluster following the $L_1 - norm$ in a first stage. Then, in a second stage, each micro cluster is assigned a density between three choices: low, medium or high. This density is used to output the final clusters. The final clusters are comprised of connected high density micro clusters for the inside and either medium or high density micro clusters for the border. The low density micro clusters may be considered as outliers. From these clustering results, it has be shown in [Barbosa et al., 2017] that a Timed Automaton could be retrieved to model the system, using a parallel between the clusters created and the states of the automaton. Based on this knowledge and the availability of experts to further improve the algorithm, the DyClee algorithm was chosen as a base to learn the DES structure of our hybrid model.

C. Learning continuous dynamics

Concerning the learning of continuous dynamics, the gradient descent method is probably one of the most used method to learn parameters to fit a set of data. In the health monitoring context, [Tamssaouet et al., 2020] aims at prognosing a system. Their work is based on an Inoperability Input-Output model which measures the distance between the current state of the system and the failure state. Gradient descent is used to tune the parameters of the model by comparing the inoperability estimated by the model and the one measured. However, the gradient descent method presents one major problem. As a matter of fact, in order to tune the parameters of the function, the function by itself, or at least its shape, must be known, which is not always our case. Hence, the gradient descent method is not suitable for our needs.

Neural networks are a commonly used machine learning algorithm that can be used for a wide range of purpose. They can be used to predict an output to an unknown or complex function as they are in [Djedidi and Djeziri, 2020], where a Nonlinear Autoregressive Network with eXogenous inputs (NARX) model measuring the power consumption of a phone is learned. Neural networks can be used to learn continuous dynamics but they have a high need of hyper-parameter tuning, which is a major drawback.

As explained in the first subsection, SVR can also be used to learn continuous dynamics. For example, it is used for system identification in [Drezet and Harrison, 1998] or [Zhang and Xi, 2004]. As stated in [Awad and Khanna, 2015], where more detailed information are available on SVR, this method has excellent generalization capability and does not need a complex tuning phase.

Another solution is to use Random Forest Regression (RFR) [Liaw et al., 2002]. It consists in averaging the prediction of decision trees. The data are randomly distributed among various decisions trees, which each give a prediction. All these predictions are then averaged and the obtained value is outputted. The RFR main advantages are that it is robust against overfitting as well as quite able to generalize. RFR only has two hyper-parameters to set and the obtained results are not that sensitive to them.

Looking at the related work, we propose a new methodology mixing DES structure and continuous dynamics learning techniques to obtain hybrid system models. As underlined in the DES structure learning part, the DyClee algorithm seems to be the best suited for our needs as it does not need to know the number of clusters before learning and fits dynamic processes. As for continuous dynamics, both SVR and RFR methods will be implemented and tested because they both have generalization capability and do not need a complex tuning phase.

III. Heterogeneous Petri Nets Formalism

This section describes the Heterogeneous Petri Nets (HtPN) formalism that can be used for health monitoring purposes. This formalism is extended from the Petri Nets for their ability to represent parallelism. As a matter of fact, when applying diagnosis and prognosis algorithms for system health monitoring, the ability to follow multiple tokens is necessary.

An Heterogeneous Petri Net (HtPN) is formally defined as $HtPN =< P, T, A, Guard, Jump, E, X, \Gamma, C, D, \mathbb{M}_0 >$

where:

- P is the set of places;
- T is the set of transitions;
- $A \subset (P \times T \cup T \times P)$ is the set of arcs;
- $Guard$ is the set of conditions associated with the incoming arcs of transitions;
- $Jump$ is the set of assignments associated with the outgoing arcs of transitions;
- E is the set of event labels: $E = E_o \cup E_{uo}$, where E_o is the set of observable event labels and E_{uo} is the set of unobservable event labels;
- $X \subset \mathbb{R}^{n_N}$ is the state space of the continuous state vector, where $n_N \in \mathbb{N}_+$ is the finite number of continuous state variables;
- $\Gamma \subset \mathbb{R}^{n_D}$ is the state space of the degradation state vector, where $n_D \in \mathbb{N}_+$ is the finite number of degradation state variables;
- C is the set of continuous system dynamics;
- D is the set of degradation system dynamics;
- $\mathbb{M}_0$ is the initial marking of the network.

The main subsets of an HtPN are briefly summarized hereafter. An example of an HtPN is presented in Figure 1(a). The represented machine has to empty bottles in two storages represented by tokens in p_3 and p_4. It only has one mechanism to empty the products, which is represented by p_1 when available and p_2 when busy. The bottles in p_3 and p_4 are automatically filled up. The one in p_3 is emptied when it reaches a given quantity of liquid. The one in p_4 is emptied when the user asks for it with the *empty* command. When the machine is working, if its degradation is greater than 200, the system enters a failure state represented by p_5.

A. Places and tokens

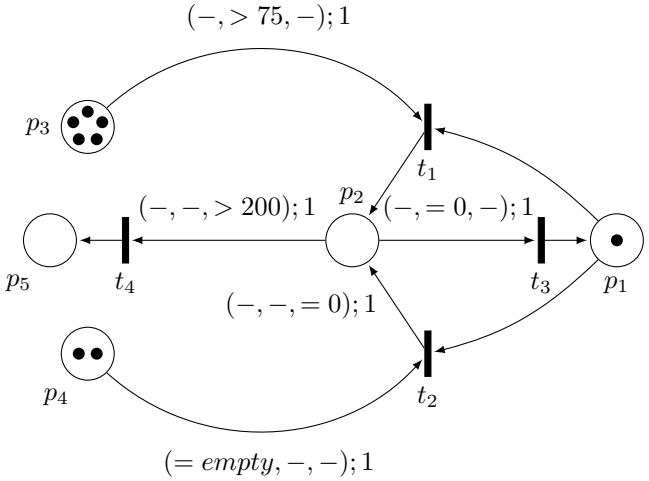
A place $p \in P$ in a HtPN is associated with a set of equations $C_p \in C$ modeling the system continuous dynamics and the associated noise (uncertainties on the evolution and on the measurements) as well as a set of equations $D_p \in D$ modeling the system degradation dynamics:

$$p = \begin{Bmatrix} C_p \\ D_p \end{Bmatrix} \tag{1}$$

The continuous dynamics C_p represents the evolution of the continuous state vector x when the system is in the place p. The system degradation dynamics D_p represents the evolution of the degradation state vector γ when the system is in the place p.

Equations associated to the places of the HtPN in Figure 1(a) are presented in Table 1(b). The symbol $-$ means that no dynamic is associated to the place. For example, places p_1 and p_5 have neither continuous nor degradation dynamics. Places p_3 and p_4 only have continuous dynamics. Place p_2, on the other hand, has both continuous and degradation dynamics.

A place p contains tokens that have three attributes: $< \delta_k^h, \pi_k^h, \phi_k^h >$, representing respectively discrete, continuous and degradation information for a token h at time k. These attributes evolve according to discrete events and dynamics associated to the place the token h belongs to.



(a)

Places	Continuous dynamics	Degradation dynamics
p_1	-	-
p_2	$x_{k+1} = x_k - 1$	$\gamma_{k+1} = \gamma_k + 3$
p_3	$x_{k+1} = x_k + 3$	-
p_4	$x_{k+1} = x_k + 2$	-
p_5	-	-

(b)

Fig. 1: Example of an HtPN (a) and dynamics associated to its places (b)

Initial marking $\mathbb{M}_0$ of the HtPN is the distribution of tokens in the different places representing the system initial conditions. Each token carries its initial discrete information (the set of events that have already occurred), its initial continuous information and its initial degradation information.

B. Arcs

The arcs of an HtPN are divided into two sets: $A = A_{\bullet t} \cup A_{t\bullet}$, where $A_{\bullet t}$ contains all the incoming arcs and $A_{t\bullet}$ contains all the outgoing arcs of transitions.

An incoming arc $a_{p,t} \in A_{\bullet t}$ going from place $p \in P$ to transition $t \in T$ is associated to a set $\Omega_{p,t} \in Guard$:

$$\Omega_{p,t} = \left\{ (\Omega^S_{p,t}, \Omega^N_{p,t}, \Omega^D_{p,t}); \rho_{p,t} \right\} \tag{2}$$

where $(\Omega^S_{p,t}, \Omega^N_{p,t}, \Omega^D_{p,t})$ represents respectively a discrete, a continuous and a degradation condition, and $\rho_{p,t} \in \mathbb{N}^+$ is the weight of the arc. For example, in Figure 1(a), the arc $a_(p_3, t_1)$ is associated to the set $\Omega_{p_3,t_1} = \left\{ (\Omega^S_{p_3,t_1}, \Omega^N_{p_3,t_1}, \Omega^D_{p_3,t_1}); \rho_{p_3,t_1} \right\}$, with:

- $\Omega^S_{p_3,t_1} = -$, which means that no discrete condition is defined,
- $\Omega^N_{p_3,t_1} => 75$, which means that the continuous state of the token in p_3 has to be greater than 75,
- $\Omega^D_{p_3,t_1} = -$, which means that no degradation condition is defined,

- $\rho_{p_3,t_1} = 1$, which means that only 1 token will be consumed during the transition firing.

A transition can be fired if the set of conditions $\Omega_{p,t}$ is satisfied for each of its input arcs. If the conditions for arcs a_{p_3,t_1} and a_{p_1,t_1} are satisfied the transition t_1 can be fired by consuming one token in p_3 and one token in p_1.

An outgoing arc $a_{t,p} \in A_{t\bullet}$ going from a transition $t \in T$ to a place $p \in P$ is associated to a set $\Omega_{t,p} \in Jump$:

$$\Omega_{t,p} = \left\{(\Omega^S_{t,p}, \Omega^N_{t,p}, \Omega^D_{t,p}); \rho_{t,p}\right\} \tag{3}$$

where $(\Omega^S_{t,p}, \Omega^N_{t,p}, \Omega^D_{t,p})$ represents respectively a discrete, a continuous and a degradation assignment and $\rho_{t,p} \in \mathbb{N}^+$ is the weight of the outgoing arc.

During the transition firing, consumed tokens are moved in the output places of the fired transition t. The attributes of these tokens remain constant or are updated. This is defined by the set of assignments $\Omega_{t,p} \in Jump$ associated with the outgoing arc. For example, in Figure 1(a), the arc a_{t_2,p_2} is associated to the set $\Omega_{t_2,p_2} = \left\{(\Omega^S_{t_2,p_2}, \Omega^N_{t_2,p_2}, \Omega^D_{t_2,p_2}); \rho_{t_2,p_2}\right\}$, with:

- $\Omega^S_{t_2,p_2} = -$, which means that the discrete information of the token will not be updated,
- $\Omega^N_{t_2,p_2} = -$, which means that the continuous state of the token will not be updated,
- $\Omega^D_{t_2,p_2} = 0$, which means the degradation state of the token going through $a_(t_2, p_2)$ will be set to 0,
- $\rho_{t_2,p_2} = 1$, which means that only 1 token will be placed in p_2 after the transition firing.

For a more detailed description of the formalism, please refer to [Vignolles et al., 2021] where the HtPN are introduced.

IV. Methodology for learning an HtPN

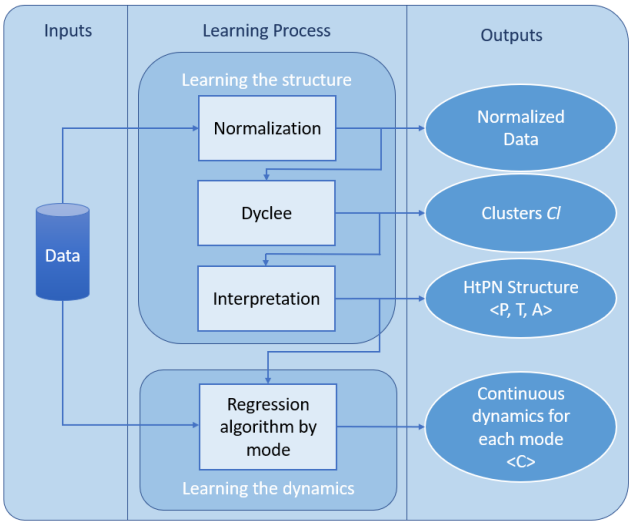


Fig. 2: Global idea for HtPN learning

The process of HtPN learning is presented in Figure 2. It requires data and is performed offline because the data normalization, the first step of the process, requires to know the maximum and the minimum values of each feature of the data. As the process is offline, the number of samples used is finite. However, if the minimum and maximum values are known beforehand, the process can be done online. Data represent the observable variables of the system, usually acquired through sensors and actuators. In the case of non-faulty sensors, data truely represent the behavior of the system. These normalized data will then be given to the Dynamic Clustering algorithm, DyClee, in a second step. The results of Dyclee are then interpreted and used to generate the structure of the HtPN. The combination of these three steps determines the number of places and transitions for the structure. A fourth step aims at learning the continuous dynamics of the system for each identified place of the obtained structure by applying regression algorithms using the data. This step outputs the continuous dynamics C associated to each place of the HtPN structure.

A. Input data

The input data used to learn the HtPN structure are the observable variables of the system. Some of these variables are sampled real values, like the water level in a tank, a pressure or a temperature. Some others are events, like the controls of a valve (open or close) or pressing a on/off button. An observable event is an event whose occurrence is detected from sensors. Input data are contained in a $i \times j$ matrix. The line i corresponds to a sample of the data. The column j corresponds to an observable variable of the data. In the case of discrete events, different events might be present on the same column, for example the open and close controls on the same valve. The set of samples is denoted S.

B. Learning the HtPN structure

1) Normalization: Raw data require normalization before the clustering process, in order for each feature to be equally important in the process. As a matter of fact, if a column of the data matrix ranges from 0 to 2 and another from 0 to 10^5, the first column might end up being neglected by the program as the values of the second column are more important. Data normalization modifies the input data file by:

- Creating a new column for each of the discrete events and converting the absence or presence of the event in a numerical value, respectively 0.0 and 1.0. This will ensure that each of the events has its own column;
- Normalizing the values with the following equation:

$$data[i,j] = \frac{data[i,j] - min[j]}{max[j] - min[j]} \tag{4}$$

with $max[j]$ and $min[j]$ respectively the maximum and the minimum values of the j^{th} column and $data[i,j]$ the value of the data of the j^{th} column and i^{th} line.

2) DyClee: Once normalized, the data are given to DyClee along with a second file containing the user parameters for DyClee process such as the desired size of the micro clusters. The size of the micro clusters needs to be chosen by an expert. DyClee outputs a set Cl of clusters, to which are assigned the data samples.

3) Interpretation of DyClee results: Clustering results obtained with DyClee are automatically processed to output an HtPN structure. The pseudo code behind the interpretation of DyClee results is presented in Algorithm 1. For each cluster $Cl_i \in Cl$, a place p_i is created.
Then, the transitions linking these places are learned.

Algorithm 1 Obtaining the HtPN structure

Input: Cl, S
Output: $HtPN\_structure : P, T, A$

$P = \{\}, T = \{\}, A = \{\}$
for all $Cl_i \in Cl$ **do**
  $P \leftarrow P \cup Create(p_i)$
end for
$act_p \leftarrow Cl^{S[0]}$
$cpt = 0$
$created\_transitions = \{\}$
for all $s \in S$ **do**
  if $Cl^s != act_p$ **then**
    $cpt += 1$
  else
    $cpt = 0$
  end if
  if $cpt == 3$ **then**
    if $act_p\_Cl^s \notin created\_transitions$ **then**
      $[T, A] \leftarrow [T, A] \cup C\_t(act_p\_Cl^s, act_p, Cl^s)$
      $created\_transitions.add(act_p\_Cl^s)$
      $act_p \leftarrow Cl^s$
    end if
  end if
end for

Let $S[0]$ be the first sample of the sample set S. The cluster in which it is affected is noted $Cl^{S[0]}$ and is saved in the variable act_p. A counter cpt which initial value is 0 and an empty set of created transitions are defined. Then, for each sample s in S, the cluster Cl^s to which it was affected by DyClee is compared with the cluster saved in act_p. As long as $Cl^s == act_p$, the value of cpt remains 0. When a sample is assigned to another cluster (*i.e.* $Cl^s != act_p$), cpt is incremented. When this counter reaches 3, clusters in act_p and Cl^s are considered linked, and a transition is created, named $act_p\_Cl^s$. The place associated to the cluster act_p is linked as the input of this transition, and the place associated to the cluster Cl^s as the output. The created transition is saved in the set $created\_transitions$ to avoid creating multiple times a same transition if the samples oscillate between two clusters. The counter cpt was implemented to avoid creating false transitions because of outliers. As a matter of fact, if an outlier is present (it is supposed that it contains less than 3 consecutive samples), the counter will prevent the creation of a transition linking the outlier with the rest of the structure, as it will not reach the transition creation threshold.

Going back to the formal definition of an HtPN, after learning the HtPN structure with DyClee, the following sets describing the HtPN structure are completed:

- P, the set of places;
- T, the set of transitions;
- $A \subset (P \times T \cup T \times P)$, the set of arcs.

C. Learning continuous dynamics

To learn the system continuous dynamics, two regression algorithms are applied: the Support Vector Regression (SVR) and the Random Forest Regression (RFR). Both algorithms are implemented using the toolbox Scikitlearn for Python. The advantage of using different algorithms is that the user is fully able to tune the learning. The user can then decide if the faster or the better fitted algorithm is chosen. To represent the fitting a coefficient of determination R^2 is used:

$$\mathrm{R}^2 = 1 - \frac{(y_{true} - y_{pred})^2}{(y_{true} - y_{true_{mean}})^2} \tag{5}$$

where y_{true} is the true value of the data and y_{pred} is the value predicted by the learned model. The closer the coefficient R^2 is to 1, the better the algorithm fits the data. It provides a measure of how well the learned model can replicate the observed outcomes. For this article, the latter choice has been made. This means that when building the final HtPN, each continuous dynamic will be represented by the learned model that better fits the data.

The continuous dynamics learning is done place by place giving the set C of continuous dynamics of the system as referenced in the formal definition of the HtPN.

The set of equations C_p associated to a place p is a fitted SVR or RFR model:

$$C_p : y_k = RM(x_k), \tag{6}$$

where RM represents the obtained regressive model and $RM(x_k)$ is the prediction given by this model for a particular continuous state vector x at time k.

Although not yet implemented, a similar process is expected when learning the set D of degradation functions. However, the challenge in learning the degradation function is the time window considered, as degradation dynamics are usually much slower than continuous dynamics describing the system functioning.

V. Application and results Results

A. System description

The benchmark used to illustrate the learning of a hybrid model is a three-tank system described in Figure 3. The data are obtained from one single simulation of the system. The *a priori* knowledge of the model is used for an easy process verification. As a matter of fact, this illustration is a case study in order to ensure that the learning process behaves as expected.

The tanks are configured in a series circuit. Flow $Q_1(t)$ delivered by the pump in tank T_1 is supposed to be constant. Tank T_2 empties with flow $Q_{20}(t)$. The available measurements at time t are the water levels h_i in each tank T_i. Valves v_{13} and v_{32} allow the water to flow between tanks. Only valve v_{13} is controlled through discrete control inputs

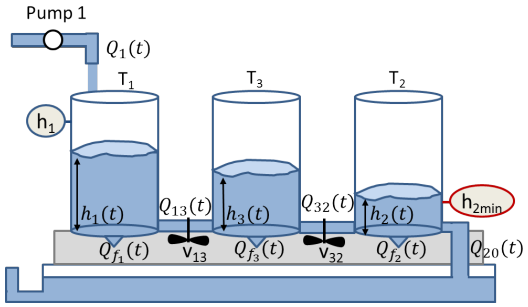


Fig. 3: Three-tank system description

$open_{v_{13}}$ and $close_{v_{13}}$. A leak may occur in tank T_1 and is represented by an unobservable fault event f_1. The goal of the system is to maintain the water level in tank T_2 greater than a minimum value $h_{2_{min}}$. The leak in tank T_1 is considered too large and therefore leads to the system failure when f_1 occurs because the system is not able to achieve the goal anymore.

The three-tank multimode representation is composed of six different modes as illustrated in Figure 4. The initial place is Nom_1, in which both valves are open. When the command $close_{v_{13}}$ occurs, the system enters mode Nom_2 in which v_{13} is closed. The system can go back to Nom_1 when $open_{v_{13}}$ occurs. From Nom_1 (resp. Nom_2), the system can switch to Deg_2 (resp. Deg_3) if fault f_1 occurs. Finally, from Deg_2 and Deg_3, the system may enter $Fail_4$ and $Fail_3$ if the water level in tank T_2 becomes smaller than the minimum value $h_{2_{min}}$. This is represented with the occurrence of a fault event f_0. The modes in the red square in Figure 4 are the *a priori* learnable modes of our system, be it directly through the occurrence of events (for $close_{v3}$ and $open_{v13}$) or through a modification in dynamics (for the occurrence of f_1).

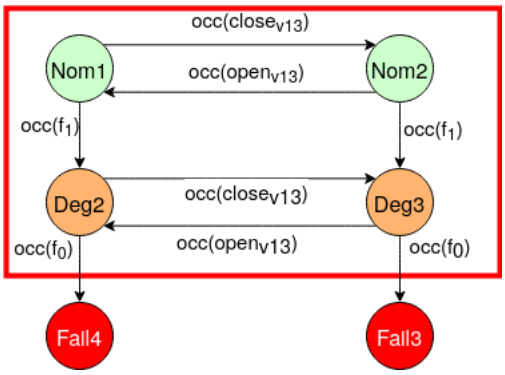


Fig. 4: Multimode representation of the three-tank system

B. Learning the HtPN structure

The simulated scenario is shown in Figure 5. 310min after the initialization, v_{13} is closed every hour during 20min in order to perform a water treatment in T_1. Fault f_1 is injected at 201840s and f_0 occurs at 206040s. Neither f_1 nor f_0

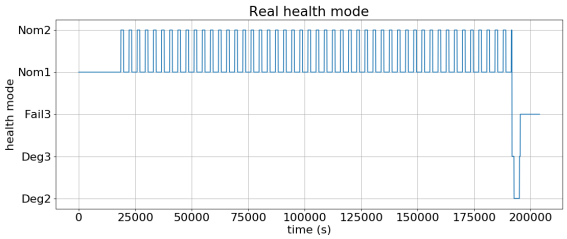


Fig. 5: The simulated scenario

are observable. Looking at this behavior, after the structural learning, five places can be expected.

A part of the data obtained with the simulated system is shown in Figure 6(a). The first column is the time stamp of the sample. The second, third and fourth columns are the water level in the first, second and third tank. The last column indicates the occurrences of events. When an event occurs in the data, it will be memorized until the occurrence of another event. For example, the event *openv13* occurred at $t = 0$, and is memorized until the occurrence of the event *closev13* at $t = 309$.

```
308|2.0979920734557616|1.8823981584084746|1.98808594503705|openv13
309|2.1323807007763182|1.8827729021376856|1.954482648376446|closev13
```

(a)

```
308.0, 0.3235, 0.8272, 0.8217, 1.0, 0.0
309.0, 0.3314, 0.8274, 0.8078, 0.0, 1.0
```

(b)

Fig. 6: Example of data before (a) and after (b) normalization

For illustration, the data given in Figure 6(a) are shown post normalization in Figure 6(b). The first column remains unchanged as it corresponds to the time stamps of data. The values in columns 2, 3 and 4 were normalized between 0 and 1. The last column is split in two columns: one for the event *openv13* and one for the event *closev13*. Each of these columns takes the value 1.0 when the corresponding event has occurred and 0.0 otherwise.

The normalized data are used by DyClee configured with a chosen size of the micro clusters equals to 0.2. This value was obtained empirically.

The results obtained by DyClee are shown in Figure 7: 4 clusters are obtained.

The results from DyClee are then interpreted to learn the HtPN structure. For each of the four clusters identified by DyClee, a place p_i is created, with $i \in [1..4]$. Five transitions are created by using Algorithm 1: $p_1 \rightarrow p_2$, $p_2 \rightarrow p_1$, $p_1 \rightarrow$

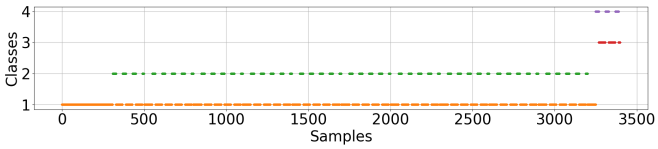


Fig. 7: Clusters identified by Dyclee

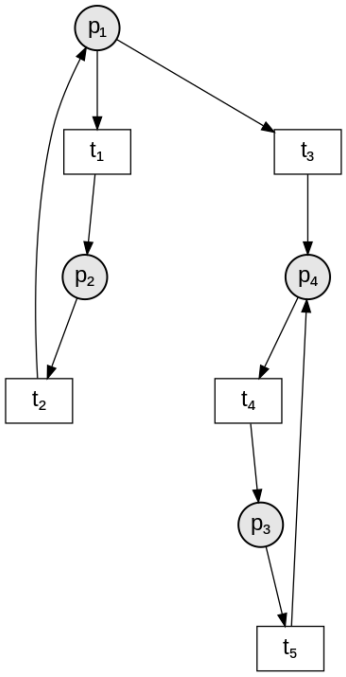


Fig. 8: Obtained HtPN structure

p_4, $p_4 \rightarrow p_3$ and $p_3 \rightarrow p_4$. Figure 8 illustrates the learned HtPN structure with places and transitions.

The clusters identified by DyClee can be compared to the true modes of the simulated system illustrated in Figure 5. First, looking at both figures, mode Nom_1 can be linked to place p_1 and mode Nom_2 can be linked to place p_2. The injection of fault f_1 can be noticed in Figure 7 as the change to places p_3 and p_4. As a matter of fact, although being unobservable, fault f_1 impacts the behavior of the water level in tank T_1. However, the fault f_0 goes unnoticed as fault f_0 represents the failure of the system and is based on the water level in tank T_2 crossing a given threshold. Fault f_0 does not impact any dynamics of the system, thus it cannot be detected by DyClee.

The whole multimode representation could not be learned from observations in the available data. As said earlier, the learned part corresponds to the red square in Figure 4.

Considering these comments, the learning of the HtPN structure performed by Dyclee gives satisfying results. It is able to model the behavior of the system quite precisely, as we managed to recover all the multimode representation that was retrievable with the available data.

C. Learning continuous dynamics

During the complete learning process, all the continuous dynamics associated with the continuous variables are learned. For understandability purposes, we illustrate the learning process only on three specific sets of samples for the water level in tank T_1 by applying RFR and SVR algorithms. The first dataset contains samples from 0 to 309 which correspond to the place p_1 from the operation start until the occurrence of event $close_{v13}$. The second set contains samples from 310 to 329 associated to place p_2. The last dataset contains samples from 330 to 370 for which the system returns in place p_1. The results of the dynamics learning process for the three datasets are shown respectively in Figures 9, 10 and 11. The real data are represented with red dots, the fitted SVR with the blue line and the fitted RFR with the yellow line. For each mode (*i.e.* a place), the fitted algorithms are scored using the coefficient of determination R^2 and the computational time and are compared in Table I.

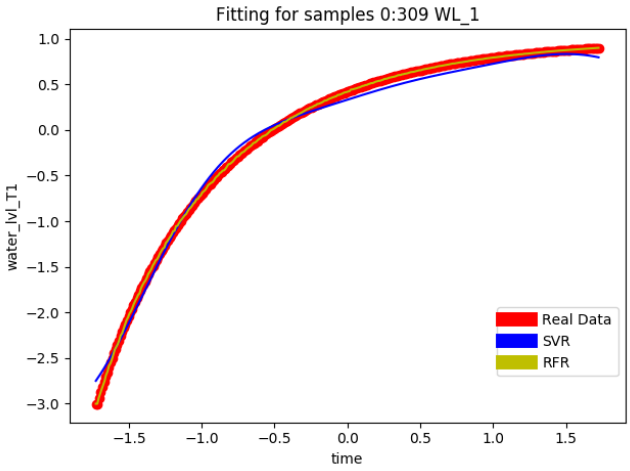


Fig. 9: Algorithms results for samples [0:309]

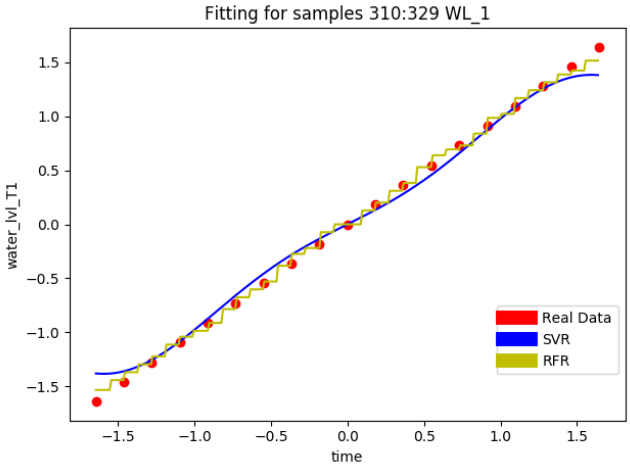


Fig. 10: Algorithms results for samples [310:329]

	R^2		fitting time	
samples	SVR	RFR	SVR	RFR
0 to 309	0.99356	0.99997	0.7ms	6.7ms
310 to 329	0.98908	0.99604	0.4ms	4.4ms
330 to 370	0.68164	0.99003	0.3ms	4.4ms

TABLE I: Comparison of the R^2 score and fitting time

For this particular application, the SVR is usually faster

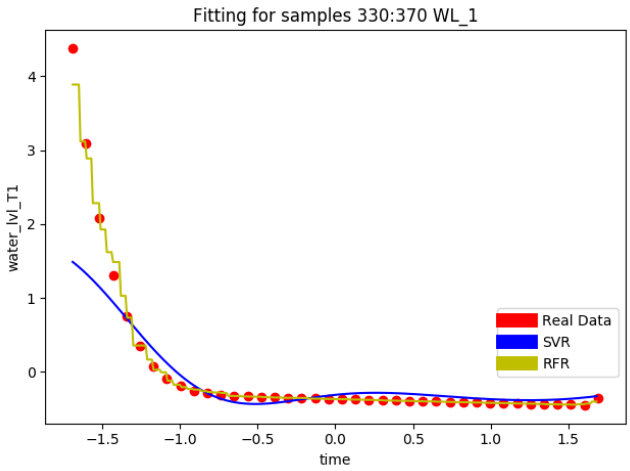


Fig. 11: Algorithms results for samples [330:370]

by a factor 10 but has a very slightly inferior R^2 to the one of RFR, except for the samples 330 to 370 for which its R^2 shows that the fitted model cannot really convey the dynamics of the system.

It can also be noticed that the continuous dynamic for h_1 learned from samples 0 to 309 is very different from the continuous dynamics learned from samples 330 to 370, even if it is considered as the same place p_1. This can be explained by the fact that samples 0 to 309 can be associated with an initialization phase in the system behavior. Considering this, an additional place could be added in the set P of the HtPN model and associated with the samples identified as the initialization phase in the system behavior. In this case, a place p_0 and a transition from p_0 to p_1 are created.

VI. Conclusion

This article presents a method to learn hybrid model for system monitoring from data.

The HtPN formalism is chosen to perform hybrid system health monitoring under uncertainty because of its capability to represent the evolution of different hypotheses on the hybrid system health state thanks to parallelism. However, the learning process could be applied to other formalisms for hybrid systems like hybrid automata. The first step of the learning method focuses on the global HtPN structure comprised of a set of places P, a set of transitions T and a set of arcs A using the DyClee clustering method. The second step aims at learning continuous dynamics associated to each place of the obtained structure. Continuous dynamics C are learned through RFR and SVR regression algorithms. The proposed learning method was applied on a three tank example. Based on simulation data, a hybrid model was learned and compared to the *a priori* known model. The results obtained are satisfying as the learned model fits what was expected given the available data.

Future work will focus on the learning of the missing parts of the model, as well as the possibility to learn and improve a known model. Although missing, the sets E, X and $\mathbb{M}_0$ can be easily retrieved from the inputted raw data and are thus considered easy to obtain. The retrieval of the sets $Guard$, $Jump$, Γ, and D, however, will be the core of our future works as it appears to require thorough works. Moreover, the learnability of the process has to be defined and thoroughly investigated.

References

[Awad and Khanna, 2015] Awad, M. and Khanna, R. (2015). Support vector regression. In *Efficient learning machines*, pages 67–80. Springer.

[Barbosa et al., 2017] Barbosa, N. A., Travé-Massuyès, L., and Grisales, V. H. (2017). Diagnosability improvement of dynamic clustering through automatic learning of discrete event models. *IFAC-PapersOnLine*, 50(1):1037–1042.

[Djedidi and Djeziri, 2020] Djedidi, O. and Djeziri, M. A. (2020). Power profiling and monitoring in embedded systems: A comparative study and a novel methodology based on narx neural networks. *Journal of Systems Architecture*, 111:101805.

[Dotoli et al., 2008] Dotoli, M., Fanti, M. P., and Mangini, A. M. (2008). Real time identification of discrete event systems using petri nets. *Automatica*, 44(5):1209–1219.

[Drezet and Harrison, 1998] Drezet, P. and Harrison, R. (1998). Support vector machines for system identification.

[Feng et al., 2010] Feng, C., Lagoa, C. M., and Sznaier, M. (2010). Hybrid system identification via sparse polynomial optimization. In *Proceedings of the 2010 American Control Conference*, pages 160–165. IEEE.

[Henzinger, 2000] Henzinger, T. (2000). The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer.

[Lauer and Bloch, 2008] Lauer, F. and Bloch, G. (2008). Switched and piecewise nonlinear hybrid system identification. In *HSCC 2008*, pages 330–343. Springer.

[Liaw et al., 2002] Liaw, A., Wiener, M., et al. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

[Moreira and Lesage, 2019] Moreira, M. V. and Lesage, J.-J. (2019). Discrete event system identification with the aim of fault detection. *Discrete Event Dynamic Systems*, 29(2):191–209.

[Pillonetto, 2016] Pillonetto, G. (2016). A new kernel-based approach to hybrid system identification. *Automatica*, 70:21–31.

[Roa et al., 2019] Roa, N. B., Travé-Massuyès, L., and Grisales-Palacio, V. H. (2019). Dyclee: Dynamic clustering for tracking evolving environments. *Pattern Recognition*, 94:162–186.

[Tamssaouet et al., 2020] Tamssaouet, F., Nguyen, K. T., Medjaher, K., and Orchard, M. (2020). A contribution to online system-level prognostics based on adaptive degradation models. In *ePHM Conf.*, volume 5.

[Vignolles et al., 2021] Vignolles, A., Chanthery, E., and Ribot, P. (2021). Modeling complex systems with Heterogeneous Petri Nets (HtPN). working paper or preprint.

[Zhang and Xi, 2004] Zhang, L. and Xi, Y. (2004). Nonlinear system identification based on an improved support vector regression estimator. In *International Symposium on Neural Networks*, pages 586–591. Springer.